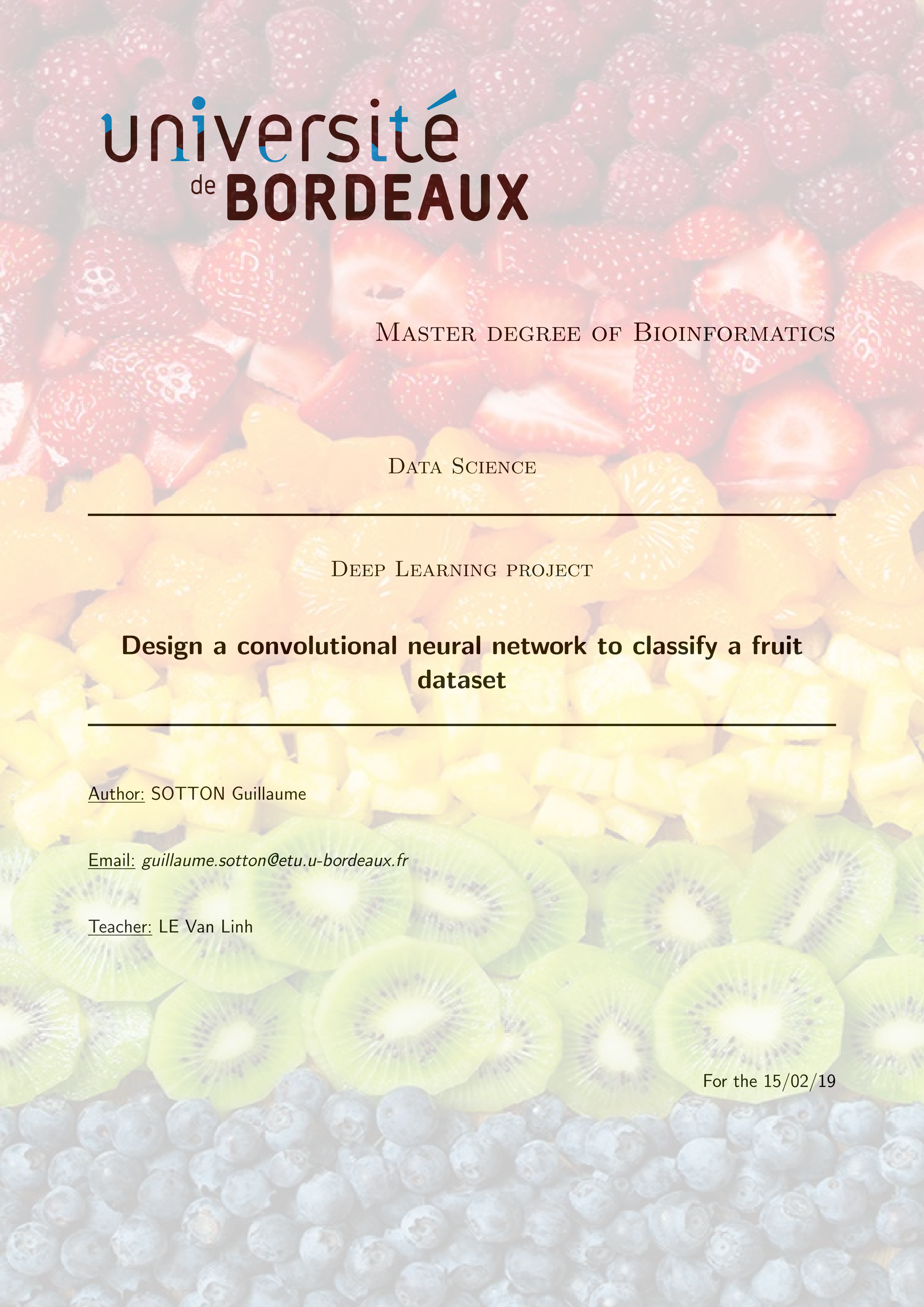


Master degree of Bioinformatics

Data Science

Deep Learning project

Design a convolutional neural network to classify a fruit dataset

Author: SOTTON Guillaume

Email: *guillaume.sotton@etu.u-bordeaux.fr*

Teacher: LE Van Linh

For the 15/02/19

1 Introduction

Deep Learning is a subfield of machine learning that teaches computers to do what comes naturally to humans: learn by example. In deep learning, a computer model learns to perform classification tasks directly from text, images or sound for example. Deep learning models can achieve accuracy that sometimes are much better than human-level performance. Models are trained by using a large set of labeled data and neural network architectures. These neurals networks contain many layers. The figure 1 below illustrates an example of neural network with different layers.

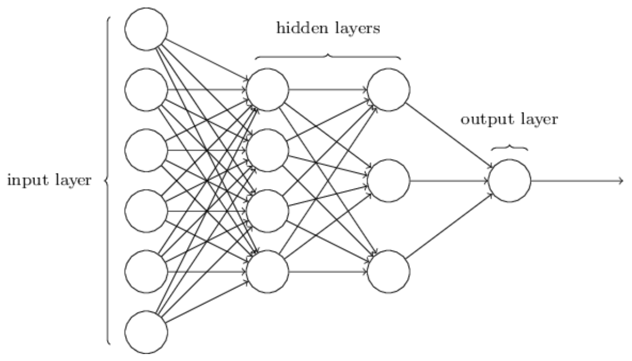


Figure 1: Example of neural network with the different layers

Deep learning is used in many domains like:

- Driverless cars
- Smartphones such as Siri on iPhone
- Audio recognition

In this study, convolutional neural networks or CNN are the models of interest. In fact, CNN is a class of deep neural networks that is commonly applied to analyze visual images. Contrary to neural networks, CNN are composed of convolutional layers, pooling layers, Rectied LinearUnit(ReLU) layers, fully connected layers and loss layers. In most cases of CNN architecture, each convolutional layer is followed by a ReLU layer, then a Pooling layer. This pattern can be repeated several times. Finally, the CNN ends by one or more fully connected layer. The figure 2 below illustrates an example of convolutional neural network with the different layers evoked just before.

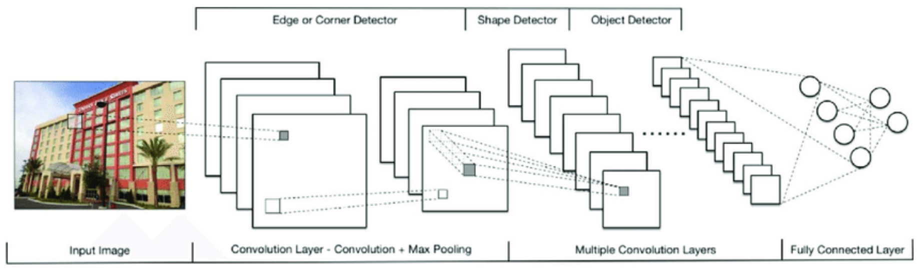


Figure 2: Example of convolutional neural network with the different layers evoked above

As part of our second year bio-informatics master's degree program, we are responsible for designing a Convolutional Neural Network (CNN) implemented in a Python program to classify the fruits in a specific dataset. This program is based on a starting file containing a set of fruit collected by *Horea Muresan* and *Mihai Oltean*[1] in an experiment studying the training of a neural network that detect fruits.

The program is available on the clickable GitHub link below:
https://github.com/GuillaumeSotton/CNN_Fruits-360

1.1 Objectives

In this study, the purpose is to design a program that does convolutional neural network on a specific dataset. It will be important to:

- Propose a CNN architecture with the purpose of classifying images of the Fruits-360 dataset,
- To implement the CNN with the framework PyTorch,
- To train and test our model and evaluate the accuracy of the latter.

2 Materials and methods

In this section are identified the various software packages, datasets and the methods that have been used for this study.

2.1 Materials

2.1.1 Dataset used

For this study, the data used is the dataset Fruits-360. It was created by Horea Muresan and Mihai Oltean in an experiment studying the training of a neural network that detect fruits. These data were created by filming the fruits while they are rotated by a motor and then extracting frames concomitantly. The fruits were placed behind a white sheet of paper as background. Fruits were scaled to fit a 100x100 pixels image. The result dataset has 57 167 images of fruits spread across 83 labels. Is shown in the annex (Table 1, a complete table of the dataset. This table was made because as it says in the article[1], the dataset evolves day after day and new labels are added. In comparison of the article provides, our dataset contains 83 labels instead of 81 in the article. The two additional labels are Grape Blue and Redcurrant.

By looking more closely the dataset, we can clearly see that the "Training" folder has a lot more files than the "Test" folder. We count 42798 pictures for the training folder, so approximately a mean of 515 pictures per labels. We count 14369 pictures for the testing folder, so approximately a mean of 173 pictures per labels.

The updated dataset is available on the clickable Github link below:
https://github.com/Horea94/Fruit-Images-Dataset

2.1.2 Programming language and framework used

During this project, the language *Python* in its version (v. 3.7.2) was used. Simultaneously, the open-source machine learning library for *Python*, PyTorch was used in its last version with the 9.0 version of CUDA.

PyTorch was used because it has several advantages such as:

- PyTorch is based on *Python*. Furthermore, *Python* is the most popular coding language in deep learning,
- PyTorch has one of the most faster deep learning training,
- PyTorch is very simple to use, learn and simple to code for the beginner.

2.2 Methods

2.2.1 Convolutional neural network designed

For this project, we used a CNN as described in introduction of this report. This type of network enables the use of:

- Convolutional layers,
- Pooling layers
- ReLU layers
- Fully connected layers
- Loss layer

It is important to notice that CNN takes into account the structure of the images while processing them. The input images that we used were RGB images of size 100x100 pixels.

The figure 3 below illustrate the CNN model that was built for this project:

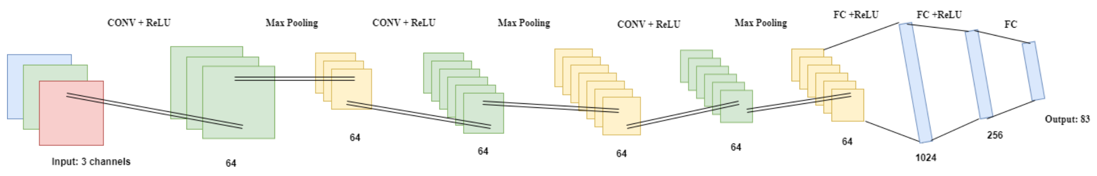


Figure 3: Convolutional neural network designed and that will be tested against the Fruits-360 dataset

As we see it above, our model used:

-A first layer that is a convolutional layer that applied 64 5x5 filters. On this layer was applied a pooling layers with a filter size of 2x2. Both of these layers had a stride equal to one.

-The second convolutional layer applied 64 7x7 filters. On this layer was applied a pooling layers as above with a filter size of 3x3. Both of these layers had also a stride of one.

-The third and last convolutional layer applied 64 7x7 filters. After that, a pooling layers was applied with a filter size of 5x5. The stride applied on these two layers was the same that we saw above.

-Then the fourth layer was a fully connected layer which had a size of 5x5.

-This layer fed into another fully connected layer with 1024 inputs and 256 outputs.

-Finally, the last layer was a softmax loss layer with 256 inputs. The number of outputs that was 83 was equal to the number of classes.

Some of these values were chosen according to the model presented in the article given in the dataset folder[1].

2.2.2 Test of the convolutional neural network model

After designing the CNN model, it was important to define some parameters with the purpose of performing our program. So, it was chosen to play on some parameters such as:

- **The batch size** which is the total number of training examples present in a single batch,
- **Number of epochs** which is when the entire dataset is passed forward and backward through the neural network. It is done x time according to the number specify for this parameter,
- **Stride**,
- **Filter size**,
- **The optimizer** parameters such as the learning rate.

For testing our model with these different parameters, the CUDA was used to set up and run CUDA operations. It keeps track of the currently selected GPU, and all CUDA tensors allocated were by default created on that device. In fact, CUDA allowed users to perform their program much faster than if it was the CPU. Thus, if the CNN model is very complex and the different parameters selected have very high values, it is an obligation to run the program on the GPU.

After that, the following parameters were chosen to test the model that we saw above in Figure 3 :

- The batch size = 200 for the training set and 100 for the test set,
- Number of epochs = 20,
- Stride = 1,
- Filter size = 5x5 on the first convolutional layer, then 7x7 on the second and the last one,
- The optimizer parameters such as lr = 0.01 and momentum = 0.9

3 Results

During our test, the following fruits images were picked by the program as test set :

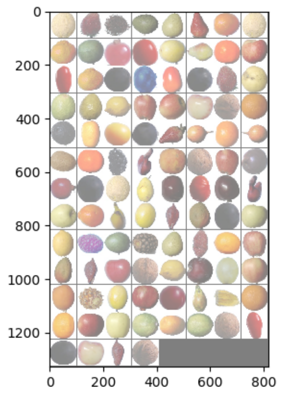
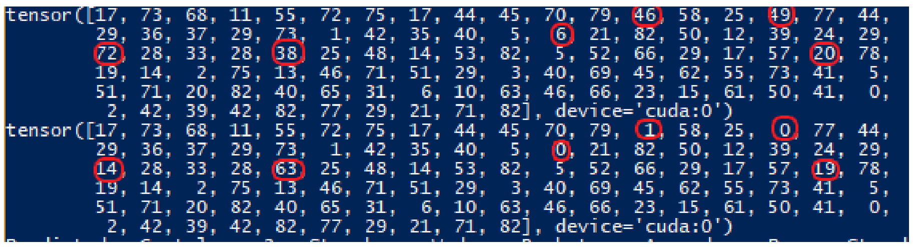


Figure 4: Fruits images picked during the program execution

Thanks to the CNN model built, the program *project.py* was performed. We saw previously the parameters chosen for this trial and they allowed us to get an accuracy of 94%. Furthermore, we saw that during the processing of the twenty epochs, the loss decrease from 4.42 to 0.0046.

The labels predicted by our model are available on the table 2. Thanks to this latter, we can see that our model is reliable and can predict very accurately the fruits labels of the test set. Furthermore, we see in red the labels that have not been correctly predicted, there are six among one hundred. So it explains the accuracy of 94%.

The tensors obtained after execution of the program are the following :



```
tensor([17, 73, 68, 11, 55, 72, 75, 17, 44, 45, 70, 79, 46, 58, 25, 49, 77, 44,
        29, 36, 37, 29, 73,  1, 42, 35, 40,  5,  6, 21, 82, 50, 12, 39, 24, 29,
        72, 28, 33, 28, 38, 25, 48, 14, 53, 82,  5, 52, 66, 29, 17, 57, 20, 78,
        19, 14,  2, 75, 13, 46, 71, 51, 29,  3, 40, 69, 45, 62, 55, 73, 41,  5,
        51, 71, 20, 82, 40, 65, 31,  6, 10, 63, 46, 66, 23, 15, 61, 50, 41,  0,
         2, 42, 39, 42, 82, 77, 29, 21, 71, 82], device='cuda:0')
tensor([17, 73, 68, 11, 55, 72, 75, 17, 44, 45, 70, 79,  1, 58, 25,  0, 77, 44,
        29, 36, 37, 29, 73,  1, 42, 35, 40,  5,  0, 21, 82, 50, 12, 39, 24, 29,
        14, 28, 33, 28, 63, 25, 48, 14, 53, 82,  5, 52, 66, 29, 17, 57, 19, 78,
        19, 14,  2, 75, 13, 46, 71, 51, 29,  3, 40, 69, 45, 62, 55, 73, 41,  5,
        51, 71, 20, 82, 40, 65, 31,  6, 10, 63, 46, 66, 23, 15, 61, 50, 41,  0,
         2, 42, 39, 42, 82, 77, 29, 21, 71, 82], device='cuda:0')
```

Figure 5: Obtained tensors after execution of the program

We can see surrounded in red, tensors that do not have the same value. The mismatched values corroborate perfectly with the prediction table 2.

Furthermore, in this project was asked to perform our program with only one image as test image. The accuracy of our program with only one image as test image(a Lime in our case) was 100%. It was decided for this project to give the program with a batch size of the test set equals to one hundred for the sake of precision.

However, during this project several CNN models and summary tables were built. Their configurations are viewable in the annexes part on the table 3. These different models where built according to the example of the article of Horea Muresan and *Mihai Oltean*[1]. In fact, in this article, we can read that the authors have tested a lot of CNN models before making their choices on which CNN was the best. That's why I decided to use the same principle for my project.

4 Conclusion

In this study was built a CNN architecture that had the purpose of classifying images from the Fruits-360 dataset. A program *project.py* was designed and implemented with the *Python* framework PyTorch. After having selected appropriate settings, the program gave us an accuracy of 94% for predicting fruits labels that were present in the test set. Furthermore, during this project several CNN models were built, the majority of them gave pretty good accuracy but some of them were not retained because their execution time were too long and their accuracies were not as good as others models. Also, we saw during this project that the building of the CNN model and the the choice of parameters values are very important. As prospects, it could be interesting to test this model on the updated dataset which count more labels than this dataset.

References

[1] Horea Mureşan and Mihai Oltean. Fruit recognition from images using deep learning. *Acta Universitatis Sapientiae, Informatica*, 10(1):26–42, 2018.

[2] Wikipedia. Deep Learning. https://en.wikipedia.org/wiki/Deep$_l$*earning*.

[3] PyTorch. Get Started with PyTorch. https://pytorch.org/get-started/locally/.

5 Annexes

Label	Number of training images	Number of test images
Apple_Braeburn	492	164
Apple_Golden_1	492	164
Apple_Golden_2	492	164
Apple_Golden_3	481	161
Apple_Granny_Smith	492	164
Apple_Red_1	492	164
Apple_Red_2	492	164
Apple_Red_3	429	144
Apple_Red_Delicious	490	166
Apple_Red_Yellow	492	164
Apricot	492	164
Avocado	427	143
Avocado_ripe	491	166
Banana	490	166
Banana_Red	490	166
Cactus_Fruit	490	166
Cantaloupe_1	492	164
Cantaloupe_2	492	164
Carambula	490	166
Cherry_1	492	164
Cherry_2	738	246
Cherry_Rainier	738	246
Cherry_Wax_Black	492	164
Cherry_Wax_Red	492	164
Cherry_Wax_Yellow	492	164
Clementine	490	166
Cocos	490	166
Dates	490	166
Granadilla	490	166
Grape_Blue	984	328
Grape_Pink	492	164
Grape_White	490	166
Grape_White_2	490	166
Grapefruit_Pink	490	166
Grapefruit_White	492	164
Guava	490	166
Huckleberry	490	166
Kaki	490	166
Kiwi	490	156
Kumquats	490	166
Lemon	466	164
Lemon_Meyer	490	166
Limes	492	166
Lychee	490	166

Label	Number of training images	Number of test images
Mandarine	490	166
Mango	490	166
Maracuja	490	166
Melon_Piel_de_Sapo	738	246
Mulberry	492	164
Nectarine	492	164
Orange	479	160
Papaya	492	164
Passion_Fruit	490	166
Peach	492	164
Peach_Flat	492	164
Pear	492	164
Pear_Abate	490	166
Pear_Monster	490	166
Pear_Williams	490	166
Pepino	490	166
Physalis	492	164
Physalis_with_Husk	492	164
Pineapple	490	166
Pineapple_Mini	493	163
Pitahaya_Red	490	166
Plum	447	151
Pomegranate	492	164
Quince	490	166
Rambutan	492	164
Raspberry	490	166
Redcurrant	492	164
Salak	490	162
Strawberry	492	164
Strawberry_Wedge	738	246
Tamarillo	490	166
Tangelo	490	166
Tomato_1	738	246
Tomato_2	672	225
Tomato_3	738	246
Tomato_4	479	160
Tomato_Cherry_Red	492	164
Tomato_Maroon	367	127
Walnut	735	249
Total	**42798**	**14369**

Table 1: Detailed table of Fruits-360 dataset

Fruits labels from the test set	Labels predicted
Cantaloupe_2	Cantaloupe_2
Strawberry_Wedge	Strawberry_Wedge
Rambutan	Rambutan
Avocado	Avocado
Pear	Pear
Strawberry	Strawberry
Tangelo	Tangelo
Cantaloupe_2	Cantaloupe_2
Mandarine	Mandarine
Mango	Mango
Redcurrant	Redcurrant
Tomato_4	Tomato_4
Maracuja	**Apple_Golden_1**
Pear_Williams	Pear_Williams
Clementine	Clementine
Nectarine	**Apple_Braeburn**
Tomato_2	Tomato_2
Mandarine_1	Mandarine_1
Grape_Blue	Grape_Blue
Huckleberry	Huckleberry
Kaki	Kaki
Grape_Blue	Grape_Blue
Strawberry_Wedge	Strawberry_Wedge
Apple_Golden_1	Apple_Golden_1
Limes	Limes
Guava	Guava
Lemon	Lemon
Apple_Red_1	Apple_Red_1
Apple_Red_2	**Apple_Braeburn**
Cherry_Rainier	Cherry_Rainier
Walnut	Walnut
Orange	Orange
Avocado_ripe	Avocado_ripe
Kumquats	Kumquats
Cherry_Wax_Yellow	Cherry_Wax_Yellow
Grape_Blue	Grape_Blue
Strawberry	**Banana_Red**
Granadilla	Granadilla
Grapefruit_Pink	Grapefruit_Pink
Granadilla	Granadilla
Kiwi	**Pineapple_Mini**
Clementine	Clementine
Mulberry	Mulberry
Banana_Red	Banana_Red
Peach	Peach
Walnut	Walnut
Apple_Red_1	Apple_Red_1
Passion_Fruit	Passion_Fruit
Pomegranate	Pomegranate

Grape_Blue	Grape_Blue
Cantaloupe_2	Cantaloupe_2
Pear_Monster	Pear_Monster
Cherry_2	**Cherry_1**
Tomato_3	Tomato_3
Cherry_1	Cherry_1
Banana_Red	Banana_Red
Apple_Golden_2	Apple_Golden_2
Tangelo	Tangelo
Banana	Banana
Maracuja	Maracuja
Salak	Salak
Papaya	Papaya
Grape_Blue	Grape_Blue
Apple_Golden_3	Apple_Golden_3
Lemon	Lemon
Raspberry	Raspberry
Mango	Mango
Pineapple	Pineapple
Pear	Pear
Strawberry_Wedge	Strawberry_Wedge
Lemon_Meyer	Lemon_Meyer
Apple_Red_1	Apple_Red_1
Papaya	Papaya
Salak	Salak
Cherry_2	Cherry_2
Walnut	Walnut
Lemon	Lemon
Plum	Plum
Grape_White	Grape_White
Apple_Red_2	Apple_Red_2
Apricot	Apricot
Pineapple_Mini	Pineapple_Mini
Maracuja	Maracuja
Pomegranate	Pomegranate
Cherry_Wax_Red	Cherry_Wax_Red
Cactus_Fruit	Cactus_Fruit
Physalis_with_Husk	Physalis_with_Husk
Orange	Orange
Lemon_Meyer	Lemon_Meyer
Apple_Braeburn	Apple_Braeburn
Apple_Golden_2	Apple_Golden_2
Limes	Limes
Kumquats	Kumquats
Limes	Limes
Walnut	Walnut
Tomato_2	Tomato_2
Grape_Blue	Grape_Blue
Cherry_Rainier	Cherry_Rainier
Salak	Salak
Walnut	Walnut

Table 2: Fruits labels predicted by the CNN model tested

Configuration			Accuracy on test set
Convolutional	5x5	6	87%
Convolutional	5x5	16	
Fully connected	-	120	
Fully connected	-	84	
Batch size	-	4	
Epochs	-	2	
Convolutional	5x5	64	91%
Convolutional	7x7	64	
Convolutional	7x7	64	
Fully connected	-	1024	
Fully connected	-	256	
Batch size	-	100	
Epochs	-	5	
Convolutional	5x5	64	94%
Convolutional	7x7	64	
Convolutional	7x7	64	
Fully connected	-	1024	
Fully connected	-	256	
Batch size	-	100	
Epochs	-	20	
Convolutional	5x5	64	83%
Convolutional	7x7	64	
Convolutional	7x7	64	
Fully connected	-	120	
Fully connected	-	84	
Batch size	-	10	
Epochs	-	10	
Convolutional	5x5	64	85%
Convolutional	7x7	64	
Convolutional	7x7	64	
Fully connected	-	120	
Fully connected	-	84	
Batch size	-	50	
Epochs	-	30	
Convolutional	5x5	256	87%
Convolutional	7x7	256	
Convolutional	7x7	256	
Fully connected	-	1024	
Fully connected	-	256	
Batch size	-	4	
Epochs	-	2	
Convolutional	5x5	256	92%
Convolutional	7x7	256	
Convolutional	7x7	256	
Fully connected	-	1024	
Fully connected	-	256	
Batch size	-	10	
Epochs	-	10	

Table 3: Accuracy on different CNN models tested