Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Classification of common fruits using neural network

Authors:
F. Ierardi – 861937 – f.ierardi1@campus.unimib.com
L. Ravazzi – 852646 – l.ravazzi@campus.unimib.com
S. Tamburini – 813117 – s.tamburini1@campus.unimib.com

June 4, 2021

Abstract

Fruit classification is a demanding task due to the large amount of types and varieties of fruit that exists, some of which can appear very similar. In this report, an approach that compares three models of different complexity is proposed, in order to find out the best one on a simple dataset with only one fruit per image. For this purpose, three different architectures were used: fully connected neural network, convolutional neural network built from scratch and convolutional neural network with fine tuning, achieving between $95\% - 98\%$ of accuracy on test set. Then, once models have been developed, they have been evaluated on a different test set containing images with more than one fruits in different contexts, with the usage of LIME as XAI technique to better understand its behavior, and the object detection task to check if focusing on regions of interests could help the model identifying more images. All these trials gave limited results, because the multiple fruits dataset proved itself to be too much different from the training set.

1 Introduction

Fruit classification performed by Machine Learning algorithms can be a very useful task for many different applications, including factory automatic fruit-packing and transportation, supermarket price determination and dietary guidance. However, the problem is difficult, because it requires distinguishing among different varieties of fruits, taking into account their relative dimensions, colors, degree of ripeness, etc. For these reasons, Neural Networks could be the best choice because they are able to learn very complex non-linear boundary functions and to automatically catch high-level features. Therefore, in this work, different types of Neural Network were implemented to recognize multiple varieties of fruit, starting from a dataset available at Kaggle.

In particular, three different architectures were implemented and trained on a dataset with a single fruit per image with a white background, described in section 2.1.

FNN Fully connected neural network (FNN) is a stack of several dense layers, in which every neuron of a layer is connected to all neurons of the previous

layers, and it was chosen here because of its simplicity and spread. In particular, it was considered interesting to test if it could reach the accuracy of more widely used models in the image classification task, like CNN, by adding more and more layers; and this hypothesis was made: this can happen if the FNN model contains a «funnel» architecture that forces the network to learn more and more abstract features at each step.

CNN from scratch Convolutional neural network (CNN), which is composed of a sequence of convolutional layers (filter banks), followed by dense layers. All the filters are trained from scratch. CNN is the most commonly used type of neural network for image classification tasks.

Fine Tuning CNN Convolutional neural network with fine tuning (Fine Tuning CNN), which uses a pre-trained convolutional net and cuts it appropriately, in order to adapt to the new provided dataset. In fine-tuning CNN filters are in part pre-trained: this is especially useful if there are not enough images to train a very complex network. In the specific case, the dataset seems big enough for the task of fruit recognition, therefore it might be possible to build a CNN from scratch with the sufficient complexity, making the use of fine-tuning unnecessary. So, are the differences in performance and learning time between CNN and fine-tuning CNN negligible? Or are the pre-trained filters of a bigger architecture useful for the classification task?

After training the three models, the performances were tested on a different dataset with multiple fruits in different contexts, described in section 2.2, using LIME as XAI technique and an object detection algorithm.

XAI and LIME Explainable Artificial Intelligence (XAI) is a relatively new technique focused on understanding and explaining why a rather complicated and black-box behaving model, like a Neural Network, gives a particular prediction, to detect if it is looking for unreliable patterns; and it was used to have an idea of the reasons why the models obtained their results. There are several XAI techniques and, in this work, *LIME* was chosen (*Local Interpretable Model-Agnostic Explanations*): it is a model-agnostic technique that firstly defines interpretable components from an instance (for example, groups of pixels called *superpixels* are used in the image case), and then finds out the function that is the best local approximation of the model, *around* that specific instance, by minimizing a *loss function* between the approxima-

tion and the whole model. Details are given in [1], while a clear mathematical explanation is given in [2]. This technique was used in order to have a clear idea of the performances of the models on the multiple fruits dataset.

Object detection The object detection task is a way of recognizing objects and their positions in a larger image; and one way of doing that is to divide the image into many Regions Of Interests (ROIs) and then apply a classifier on each of them. It was used here in order to find out if the models were usable in this task, if applied in particular to the multiple fruits dataset.

2 Datasets

The chosen dataset [3] is freely accessible on Kaggle and it contains color images about different fruits and vegetables. It can be divided in two very different part: the *single fruit* part, which will be described in section 2.1 and the *multiple fruits* part, which will be described in section 2.2.

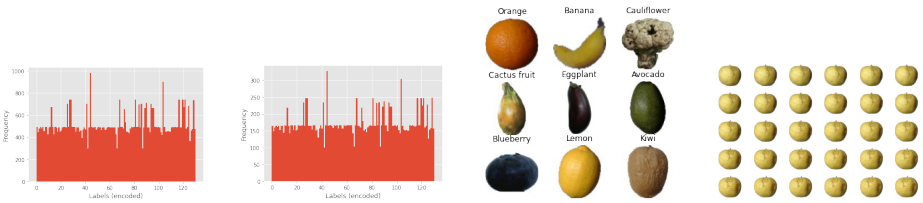
2.1 Single fruit dataset

The single fruit dataset is composed by 90.380 images, divided into a *training set* (with 67.692 images) and a *test set* (with 22.688 images). Images dimensions are 100×100 pixels, with the *RGB* color mode. Images are provided with their associated labels: different varieties are considered as different labels. There are 131 different labels, with names like *Apple Golden 1*, *Apple Red Yellow 2*, *Grape Blue* and *Nectarine*.

Labels distributions are represented in figures 1a and 1b; as it can be seen, training set and test set distributions are the same, and all classes have more or less the same number of elements (around 500), but there are some exceptions like *Grape Blue* (984), *Plum* (900) and *Ginger Root* (297), in training set.

In general, these images are unrealistic: firstly, as it is described in [3] and [4], only *one instance* of every variety is in the dataset (one specific apple Crimson Snow, one specific banana Lady Finger, etc); secondly, the fruit occupies the entire image; thirdly, the background has been digitally removed, according to the algorithm described in [4]; and finally, these photos have been generated as screencaps from a video, so different views and rotations of the same fruit are present. Some fruits are represented in image 1c, and

different views of the same fruit is sketched in 1d (an *Apple Golden 1* has been used as an example).



(a) *Training set* **(b)** *Test set* **(c)** *Different classes* **(d)** *Different views*

Figure 1 – Labels distributions and images of single-fruit dataset

2.2 Multiple fruits dataset

The multiple fruits dataset is composed by 103 *RGB* rectangular images, oriented horizontally or vertically. Image dimensions may vary: the base dimension is 3024×4032 (or 4032×3024, depending on the orientation), but sometimes they have been cut, and therefore the actual image is smaller. These images can vary a lot: they can contain the same variety multiple times, or one instance for different varieties, or different instances of different varieties; and they can be in containers like bowls, baskets and bags, or attached to their plants. However, the represented fruits are of the same types that in the single-fruit dataset. Some examples are shown in figure 2.

In this case, images are not provided with their associated labels: they are all in the same folder, without a descriptive dataset, and the images' names can help to identify the fruits represented, but the varieties usually have to be inferred. For example, sometimes there are names like *Bananas(lady_finger)1*, but usually something like *grape_pear_mandarine* (different pear varieties are present). Therefore, an explanatory dataset for this set of images has been manually generated.

3 The Methodological Approach

In this section, the three models that were developed will be described, along with the XAI model (section 3.4) and the object detection algorithm (sec-

(a) *Fruits in a bowl* **(b)** *Fruits in a basket* **(c)** *Same variety* **(d)** *Fruits and plant*

Figure 2 – Some images of the multiple fruits dataset

tion 3.5). All the models were trained using the single fruit dataset described in section 2.1: the whole training set has been used for training, and then hyperparameters have been tuned by checking the performance on a *validation set*, defined as half the test set. Finally, the other half of the test set has been used to test the final performances. Data augmentation was considered in some cases to better generalize the model to the multi-fruits dataset, and in particular flipping, rotations, zooming and width and height shifting operations.

All the code was written in Python, using in particular the Keras API (with a Tensorflow backend) for the models, because of its simplicity, user-friendliness and efficiency with Deep Learning architectures. Google Colab has been used to execute the code, because of its free GPU. All the code used will be available in a dedicated Google Drive folder.

3.1 Fully connected neural network

Firstly, the fully connected neural network model will be described here. A trial and error approach was used to train it: the first model that was developed and trained will be considered here as the *base model*, and then different hyperparameters were changed one at a time, and implemented *separately* to the base model, to better clarify and understand their impact on the learning process.

The base model To test the hypothesis formulated in section 1, a rule has been defined to generate the «funnel» architecture described there. Let X be the number of input neurons, and Y the number of output neurons. Let 2^n be the power of two closest to X (with $2^n \leq X$), e 2^m the power of

two closest to Y (with $2^m \geq Y$), with $n \geq m$ and $n, m \in \mathbb{N}$. The network will have $N = n - m + 1$ hidden layers, with r_i neurons at hidden layer h_i (with $i = 0, 1, \ldots, N - 1$), and:

$$r_i = 2^{n-i} \tag{1}$$

In the case considered, $X = 28 \times 28 \times 3 = 2352$ (image dimensions and color channels) and $Y = 131$ (number of classes), so $n = 11$ ($2^{11} = 2048$) and $m = 8$ ($2^8 = 256$); therefore the network will have $N = 4$ hidden layers: the first hidden layer will be denoted with h_0 and will have $2^{11} = 2048$ neurons; the second hidden layer, h_1, will have 1024 neurons; the third hidden layer, h_2, will have 512 neurons, and the last layer, h_3, will have 256 neurons.

Other hyperparameters chosen for the base model were: the *Xavier Uniform* weights initialization, an image resizing of 28×28 pixels, the *ReLU* activation function for hidden layers, the *softmax* activation function for output layers, no regularization techniques, the *Adam* optimizer, the categorical cross entropy loss function, and a batch of 256 images. These are standardized choices in this context, and the image resizing has been done to reduce the number of parameters as in a standard dataset like the MNIST.

This model has been trained with different epochs and learning rate, with the objective to reach smooth learning curves and the best accuracy possible without overfitting. After reaching the best combination for the base model, it has been maintained constant moving forward, and changed again only if it could significantly improve accuracy.

Changes in the base model The changes made will be briefly described below, along with their motivations.

- *Regularization*, especially the dropout technique (with different probabilities), was added, to reach even smoother curves and to better generalize on a different test set, especially considering the multi-fruits dataset. Dropout probabilities have been applied to every hidden layer, but not on input or output layers. In this case, the learning has been proven slower, and therefore more epochs were tried.

- *Data augmentation* techniques were included to train the model to recognize different views of the same image – especially considering the multi-fruits dataset.

- *Less hidden layers* were considered, to find out if accuracy will pretty much stay the same: in this case, the model would have learned sufficiently abstract features even without the need to add more layers. In particular, the hidden layers were subsequently removed from the base model: h_0 was removed first, and then h_0 and h_1 were removed, and so on, until every hidden layer was removed and only input and output layers remained.

- *More hidden layers* were added, to find out if, on the contrary, the accuracy will significantly increase: in that case, the model would have needed more levels to better learn. In particular, more hidden layers were added to the base model, in descending order in the funnel: a 2048-neurons level (between h_0 and h_1), a 128-neurons level (between h_4 and the output layer), a 1024-neurons level (between h_1 and h_2), and two hidden layers with 2048 and 1024 neurons (between h_0 and h_1, in descending order).

Regarding timing considerations, computation time scales linearly with the number of epochs, while adding or subtracting layers has a negligible impact; and data augmentation is particularly slow, especially in the FNN case: one epoch last 12 minutes, while usually it lasts around 1 minute and 20 seconds.

3.2 Convolutional neural network

The second architecture that was implemented is the convolutional neural network, built from scratch. A CNN utilizes convolutional filters that apply constraints which greatly reduce the number of trainable parameters, allowing the use of the original shape of the images as input size (instead of rescaling them) without having to create a prohibitively large network. The approach used to find the best performing model was the following: starting from a simple network composed of only convolutional layers, dense layers and regularization techniques were gradually added to the architecture. At each stage, different models with different parameter values were tested and the values that lead to the lowest loss on test set were chosen, although there are some specifics that stayed the same for all models, in particular:

- Due to the nature of the classification problem (single label, multiple classes) the loss function and the output activation functions used are,

respectively, the categorical cross-entropy and the softmax function;

- *ReLU* activation functions are used for all hidden units to avoid the "vanishing gradient" problem of sigmoid functions;
- The learning algorithm of choice was the Adam algorithm with a batch size of 128.

To compare the different models, instead of using a fixed number of epochs, an early stopping procedure has been implemented: if the loss on the validation set didn't improve for three consecutive epochs, the training was stopped and the weights from the best epoch were returned. In this way, training was stopped as the model has ceased learning or was starting to go into overfitting.

The initial architecture is composed of 4 convolutional layers, each followed by a max pooling layer which halves the width and height of the previous convolutional layer. The filter size is 3 for the first and last layers and 4 for the second and third one. This choice of filter sizes assures that the height and width of the layer before a max pooling is always an even number, so there is no loss of information after a max pooling operation even without the implementation of padding.

In the beginning, four fully convolutional models with different number of filters for the various layers have been trained and tested on the original data without augmentation. After that, to the winning architecture a single dense layer was added and three different models with different numbers of neurons for the dense layer were evaluated. The same procedure was implemented for a second dense layer.

Then, it was decided to implement some regularization techniques in order to slow down the learning on the training set and to improve the generalization capabilities of the model, hopefully resulting in higher validation and test accuracies. The regularization techniques were added only to the dense layers only: first dropout was added, then L2 weight decay as well. This type of regularization did improve the accuracy of the model, but required more training epochs.

Finally, the last model was trained once again with an early stopping patience set to 5 instead of 3. Thanks to the regularization previously implemented, it was also possible to increase the learning rate and still obtain relatively smooth loss curves, in this way the number of training epochs could stay under 50.

After having obtained the final model, it was decided to train it again implementing data augumentation on the dataset. It was notable that data augmentation slows down significantly the training process (from around 50 seconds per epoch to 3 minutes), since it generates new data every epoch and to do so it utilizes the CPU instead of the GPU.

3.3 Convolutional neural network with fine-tuning

The fine tuning approach is the last technique to carry out in order to explore different methodologies for image classification. It's preferable than feature extraction approach because there are a lot of data which allow to train several top additional layers of the net. For this purpose, *VGG16* was chosen because it's a well known architecture. It has been pre-trained with *Imagenet* dataset which is composed of millions of images tagged with 1000 labels. This choice is justified by the common usage of imagenet pre-trained weights for these tasks and due to the presence of different labels associated to fruits and vegetables in the dataset, such as apple, pomegranate, pineapples and banana. In this way, pre-trained net starts with a little help to perform the task of interest. The architecture of this net is made up of different convolutional and pooling layers and finally, a fully connected net, as sketched in figure 3.

Figure 3 – Architecture of VGG16 (from [5])

In order to use this architecture, it's mandatory to apply the *pre-processing function* which converts images into another space color, i.e. from RGB to BGR, and it computes the difference in order to zero-center each color channel with respect to the Imagenet dataset. However, this pre-processing doesn't scale the pixels between $[0, 1]$ even if it's an important step for training every neural network. Scaling step wasn't add because it isn't conceived in the original pre-processing function that define the best standard input for VGG16.

Once the pre-processing has been done, two different fine-tuning approaches were designed in order to appreciate the differences of cutting the net at different points.

First approach *Drop the last dense layers and add one or more dense layers trained by our data.* This choice is justified by the fact that some high-level features extracted from the imagenet dataset by VGG16 should adapt also to this dataset. So, it's reasonable to train only the new fully-connected layers of the model to adapt the net to the new dataset.

Firstly, the simplest possible model was implemented in order to have a performance baseline which will be improved with the introduction of different regularization techniques, such as L_1, L_2 and dropout, and adding different dense layers. In this way, by changing the complexity of the models, the trends and behavior of them can emerge.

So, let conv_base be the pre-trained architecture and output the number of labels of the fruit dataset: the simplest model was identified as conv_base + Dense(output, 'softmax'), and then, layers were added to conv_base until a reasonable threshold of complexity was achieved.

Second approach *Retrain the Conv5-1, Conv5-2, Conv5-3, i.e. the last block of convolutional layers, with provided data, drop the last dense layers and add one or more dense layers trained with fruit data.* This approach was implemented because high-level features extracted from the provided dataset could lead to more accurate predictions for the classification task than the previous ones, which didn't focus only on fruits images. Training data were considered sufficient to train also this part of the model.

The procedure implemented is the same as for the first strategy: start from the simplest method and change complexity.

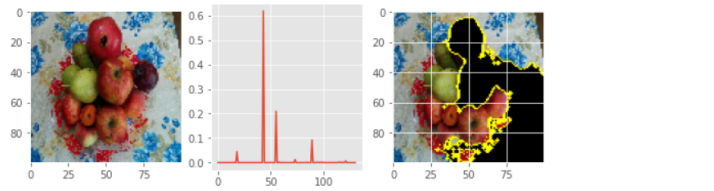
For both strategies, some models were trained with augmented images and other with the original ones in order to understand if data augmentation is useful for fruit classification. Data augmentation has applied rotation, shift, zoom and flip operations. Reasonable values of hyperparameters, i.e. learning rate, batch size and tuning values, were fixed first for all models, defining an exploration part; and then, once the best model for each strategy has been highlighted, the exploitation part occurs, in order to recognize the best set of hyperparameters to use. Early stopping wasn't used at this stage because models behaviours were compared after a fixed number of epochs.

Finally, the best model among all, regardless the strategy, was executed with the best set of hyperparameters, and early stopping was employed.

As regards the efficiency point of view, the average time for computing a single epoch is higher with data augmentation, as previously stated for FNN and CNN strategies. Especially, that computational time is independent from the complexity of the model and it scales linearly with the number of epochs.

3.4 Multiple fruits and XAI

After the three models described previously were implemented, they were tested on the multiple fruits dataset described in section 2.2. No global measure of accuracy was calculated: the models were qualitatively tested by exploring their predictions they gave image per image, and comparing them with the true values, as it can be seen from figure 4. LIME results were shown here to find out what are the parts of the image that the model considered important for its classification. Final performances were also qualitatively compared with the $VGG16$ results on the same data.



```
True label: Index(['Apple Braeburn', 'Apple Red 1', 'Apple Red 2', 'Apricot', 'Nectarine',
       'Peach', 'Peach Flat', 'Pear', 'Pear Abate', 'Pear Forelle', 'Plum',
       'Pomegranate'],
      dtype='object')    Best prediction: Granadilla

Label: Kaki                         Probability 0.20905263721942902
Label: Banana Red                   Probability 0.04465879872441292
Label: Granadilla                   Probability 0.6191798448562622
Label: Nut Pecan                    Probability 0.012439488433301449
Label: Pear Red                     Probability 0.09232771396636963
```

Figure 4 – Example of multiple-fruit testing: original image, predictions probability distribution, LIME results, true labels, best prediction and predictions with probability higher than 0.01.

3.5 Object detection

The algorithm that performs this part operates in three steps:

- It creates an *image pyramid* of the original image: a multi-scaled representation of the image obtained through repeated subsampling. This is necessary to search for objects in the image at different scales;
- It extracts the ROIs from each image of the image pyramid via a sliding window, and applies the classifier to each ROI, obtaining the output predictions: if the probability of a label is sufficiently high, label and bounding box location are saved;
- Finally, it applies non-maximum suppression to resolve overlapping bounding boxes;

For more information regarding the algorithm implementation, refer to [6].The entire process is quite slow, taking a few seconds for a single image.

4 Results and Evaluation

In this section, obtained results and numbers are presented.

FNN results Relevant results from various FNN trials are shown in table 1 in the appendix. The base model is reported in the *first row*, along with its parameters and its train, validation and test categorical accuracy; from that, the most important hyperparameters that have been *separately* changed from the base model are described in the subsequent rows, along with their best value, the number of epochs used to train it (this number is 20 most of the times, except for some particular cases), and their train, validation and test accuracy.

CNN The best models for each step of the training procedure for the training of the CNN are shown in table 2 in the appendix. After some initial trials, the learning rate was chosen to be 0.00005 for all models, except for the last two for which it was increased to 0.0001. The best results (the highest accuracy on test) were obtained with a configuration of 32 - 64 - 64 - 128 number of filters for the four layers, 1024 neurons for both dense layers, 0.5 for the parameter that specify the dropout probability and 0.005 for the tuning hyperparameter of the L2 regularization.

Fine tuning In order to explore the behavior of the two fine-tuning approaches, more than thirty models were implemented. However, only the most significance ones are reported in table 3 in the appendix. All of the models were learned with the fixed set of hyperparameters: 20 epochs, a batch size of 128 images, a learning rate of 10^{-4}, a dropout probability of 0.2, and λ parameter of 0.01 for L_1 and L_2 regularization. Moreover, the Adam optimizer, the categorical cross-entropy loss function, the categorical accuracy metrics were chosen and $ReLU$ function is used for additional dense layers. All dense layers were initialized with uniform distribution with a fix seed.

Once the best model has been identified for each strategy, the hyperparameter search occurs with batch size values within $[32, 64, 128, 256]$ and learning rate within $[10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}]$ was performed. Moreover, also the tuning parameter λ was tuned with $[0.1, 0.05, 0.01, 0.001]$ for the best second approach.

Multiple fruits test and object detection No numeric results were calculated for this part: models were evaluated only through qualitative means and will be discussed in the next section. The most relevant results are shown in figure 5 in the appendix.

5 Discussion

In this section, results are discussed separately for each part.

FNN In the FNN case, the rule defined in equation (1) has been proven useful to reach about the same levels of accuracy than CNN: as it can be seen from table 1, less layers are not enough and more layers don't improve the accuracy as significantly. The case when two hidden layers were added is particularly showing because even the train accuracy has dropped: too many layers make the model unlearn useful features. A small regularization is useful to achieve better performances on test set, especially when more epochs are added; it has to be noted though that validation and test set used here are very similar to the training set, and therefore too much regularization can get the performances slightly worse in this case. Data augmentation in this case has lowered the performances a lot, and it is too slow to even think about trying other alternatives. The best model is the base model with a small

dropout (probability 0.05), achieving 95.94% of accuracy; in general, these results are biased by the fact that the training set is composed by unrealistic images.

CNN From looking at the results of the training, it is possible to see that even with the simplest model (convolutional only) the results obtained were very good, with an accuracy on test set of 93.89%. Adding a single dense layer didn't seem to improve the performance, in fact the results obtained were slightly worse than the ones of the previous model, whereas adding two dense layers with 1024 neurons each seemed to improve the performance significantly, reaching 96.15% on test set. With those first models, it was observed that the training accuracy quickly reached values close to 100%, after which the validation accuracy tended to improve very little, although even without regularization on dense layers the results were still very good, probably due to the strong similarity between training and test set. In fact, for what concerns the regularization techniques implemented, it is worth of notice that adding dropout to the dense layers didn't improve the results significantly, whereas adding both dropout and L2 lead to slightly better results (96.93% on test). A significant improvement was obtained instead by increasing the early stopping patience to 5, showing that the model obtained at the previous step had still room for improvement. Finally, adding data augmentation lead to much worse results (90.23% on test set). This result can again be justified considering that train and test data are very similar (the images for each label are all images of the same fruit but from different angles), so adding data augmentation lead to a training dataset which was too different from the validation and test set.

Fine Tuning Analysing the result of the first approach, the simplest model has achieved a 95.87% of accuracy in the testing phase which is a good result but it can be improved: indeed, the best model for this strategy, shown in table 3, has achieved the maximum value of 96.38% of accuracy. So, it seems that some layers should be added in order to catch some characteristics of the dataset; however, it cannot be stated with certainty that the latter result is better than the former because the difference shouldn't be of statistical significance. Loss curves have been monitored in order to check if the overfitting regime starts to take place: they were decreasing and smooth. Without data augmentation the training losses achieve a 100% of accuracy

in less time, but the validation and test accuracy values remain lower rather than the cases with data augmentation: it's better to use models that gradually learn trained with augmented images. L_1 regularization technique is associated to the worst model with 85.90% of accuracy. On the other hand, L_2 regularization and Dropout seem not to have a significantly effect on performances of the models and this is reasonable: indeed, these techniques are fundamental when there are few data but it isn't the case of the single-fruit dataset. The presence of a large amount of data is the main solution for the overfitting problem solved by those methods.

As regards the second approach, the simplest model has achieved a 96.92% of accuracy which is higher than all values obtained with first approach: therefore, the second approach proves to be better than the first one. On the other hand, the best model, shown in table 3, has reached the maximum value of 98.02% of accuracy. All above mentioned comments for the first strategy remain true also for this case. Moreover, data augmentation is an useful operation because in some models it's able to reduce some discontinuities in the learning process shown by the train and validation error curves.

For both strategies, the hyperparameter search highlighted that the batch size doesn't effect too much our performances and 128 is a good compromise, while the learning rate search showed that it is able to change completely the behaviour of models and the best value is 10^{-4}. Moreover, also the λ hyperparameter has been tuned resulting that $\lambda = 0.01$ is the best one: the best set of hyperparameters was used since the beginning. The best model among all belongs to the second approach and finally, it was learned with the best set of hyperparameters and early stopping with patience equal to 5 with retrained weights achieving 97.95% of accuracy.

Multiple fruits: XAI and LIME The XAI technique proves itself useful to have an idea of what the models were doing: it was clear that what they were searching for was a single fruit, roughly the size of the entire image; and this is reasonable, looking at the training set. Therefore, in general the models performed really poorly when they had to deal with multiple fruits situated in different locations in an image. The $VGG16$ net alone performed way better on the multiple fruits dataset photos that were similar to the Imagenet dataset; and therefore Fine Tuning CNN inherited in part these capabilities: it was able to better distinguish the fruit from the rest, while not being able to assign the correct labels.

Object detection The results in this case were a mixed bag: although in general (but not always) the object detector was able to identify and localize fruits in the images, it tended to also find fruits where there were none. This might in part be due to the fact that the models didn't have a label for "not-fruit" and always tried to label everything as a fruit; but is also due to the uniform background of the training set images.

6 Conclusions

In general, FNN, CNN and CNN Fine tuning approaches performed well on the single fruit test set, with the second strategy being preferable in the fine tuning case. In particular, FNN models have slightly lower accuracy values and the maximum difference of accuracy between the best model obtained by FNN and CNN is 1.5%.

On the other hand, the difference in performance between the best models for CNN and CNN Fine Tuning is even smaller, almost equal to 0.6%, with CNN Fine Tuning achieving the highest value.

In general, regularization techniques are useful but the improvements that they bring aren't very significant. Moreover, data augmentation helps some CNN Fine Tuning models to improve performances but, on the other hand, it has penalized FNN and CNN models due to the similarity of single fruit images between training and test dataset.

However, the single fruit test set is too much like the training set, and the multiple fruit dataset is too much different; therefore the multiple fruits analysis was pretty much unsuccessful. Performances on the multiple fruit dataset could be improved by using a different and more realistic dataset, and this is supported by the better results of the $VGG16$ net on the multiple fruits images that were similar to the Imagenet ones.

References

[1] S. S. T. Ribeiro and C. Guestrin. "Why should i trust you?" (Aug. 2016), [Online]. Available: https://arxiv.org/pdf/1602.04938.pdf.

[2] P. Ferrando. "Understanding how lime explains predictions". (Dec. 2018), [Online]. Available: https://towardsdatascience.com/understanding-how-lime-explains-predictions-d404e5d1829c (visited on 05/05/2021).

[3] M. Oltean. "Fruits 360". (May 2020), [Online]. Available: https://www.kaggle.com/moltean/fruits (visited on 05/03/2021).

[4] H. Mureșan and M. Oltean, "Fruit recognition from images using deep learning", *Acta Universitatis Sapientiae, Informatica*, vol. 10, pp. 26–42, Jun. 2018. DOI: 10.2478/ausi-2018-0002.

[5] F. Li, J. Johnson, and S. Yeung. "Cnn architectures". (Apr. 2019), [Online]. Available: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture09.pdf (visited on 06/04/2021).

[6] A. Rosebrock. "Turning any cnn image classifier into an object detector with keras, tensorflow, and opencv". (Jun. 2020), [Online]. Available: https://www.pyimagesearch.com/2020/06/22/turning-any-cnn-image-classifier-into-an-object-detector-with-keras-tensorflow-and-opencv/ (visited on 05/12/2021).

A Complete list of labels

Apple Braeburn, Apple Crimson Snow, Apple Golden 1, Apple Golden 2, Apple Golden 3, Apple Granny Smith, Apple Pink Lady, Apple Red 1, Apple Red 2, Apple Red 3, Apple Red Delicious, Apple Red Yellow 1, Apple Red Yellow 2, Apricot, Avocado, Avocado ripe, Banana, Banana Lady Finger, Banana Red, Beetroot, Blueberry, Cactus fruit, Cantaloupe 1, Cantaloupe 2, Carambula, Cauliflower, Cherry 1, Cherry 2, Cherry Rainier, Cherry Wax Black, Cherry Wax Red, Cherry Wax Yellow, Chestnut, Clementine, Cocos, Corn, Corn Husk, Cucumber Ripe, Cucumber Ripe, Dates, Eggplant, Fig, Ginger Root, Granadilla, Grape Blue, Grape Pink, Grape White, Grape White 2, Grape White 3, Grape White 4, Grapefruit Pink, Grapefruit White, Guava, Hazelnut, Huckleberry, Kaki, Kiwi, Kohlrabi, Kumquats, Lemon, Lemon Meyer, Limes, Lychee, Mandarine, Mango, Mango Red, Mangostan, Maracuja, Melon Piel de Sapo, Mulberry, Nectarine, Nectarine Flat, Nut Forest, Nut Pecan, Onion Red, Onion Red Peeled, Onion White, Orange, Papaya, Passion Fruit, Peach, Peach 2, Peach Flat, Pear, Pear 2, Pear Abate, Pear Forelle, Pear Kaiser, Pear Monster, Pear Red, Pear Stone, Pear Williams, Pepino, Pepper Green, Pepper Orange, Pepper Red, Pepper Yellow, Physalis, Physalis with Husk, Pineapple, Pineapple Mini, Pitahaya Red, Plum, Plum 2, Plum 3, Pomegranate, Pomelo Sweetie, Potato Red, Potato Red Washed, Potato Sweet, Potato White, Quince, Rambutan,

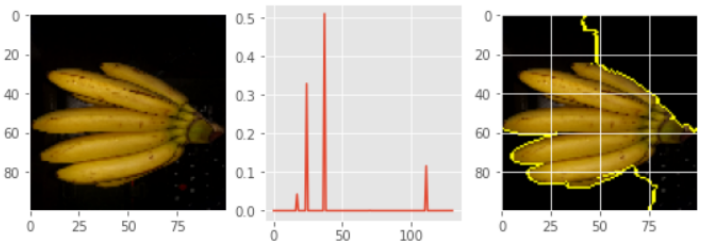
Raspberry, Redcurrant, Salak, Strawberry, Strawberry Wedge, Tamarillo, Tangelo, Tomato 1, Tomato 2, Tomato 3, Tomato 4, Tomato Cherry Red, Tomato Heart, Tomato Maroon, Tomato Yellow, Tomato not Ripened, Walnut, Watermelon.

B Selected results

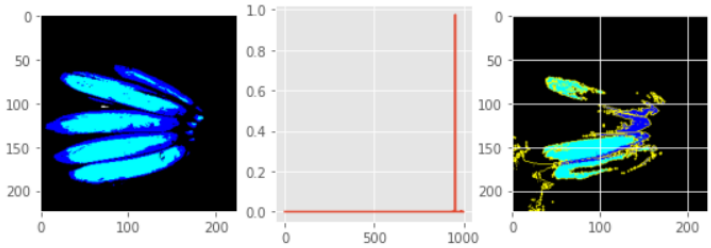
In this appendix, selected results of FNN, CNN and CNN Fine tuning trials are shown – respectively, table 1, 2 and 3. XAI and Object detection results are shown instead in figure 5.

Table 1 – Results of different FNN trials. Every change has been implemented *separately* on the base model.

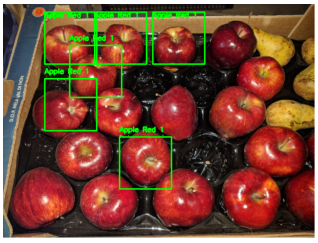
Trials	*Parameter*	*Epochs*	*Train* (%)	*Validation* (%)	*Test* (%)
Base model	Learning rate: 5×10^{-5}	20	99.98	94.72	94.15
Dropout (probability)	0.05	40	99.90	96.37	95.94
	0.3	20	96.82	93.99	93.73
Data augmentation	–	20	85.86	80.44	–
Added hidden layers	one hidden layer (2048 neurons)	20	100.00	94.57	94.17
	two hidden layers (2048 and 1024 neurons)	20	97.51	94.12	93.63
Less hidden layers	All hidden layers removed	20	93.24	78.47	77.21
	h_0 removed	20	99.94	93.81	93.45



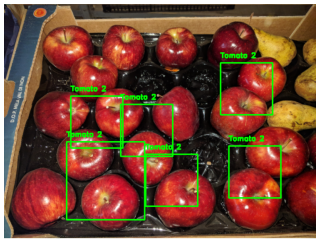
(a) *Performances of the model*



(b) *Performance of the VGG16 model*



(c) *Object detection that works*



(d) *Object detection that doesn't work*

Figure 5 – LIME and object detection results

Table 2 – The table shows the results of the best models from each step of the training procedure for the CNN architecture

Change	*Best result*	*Train acc.* (%)	*Validation acc.* (%)	*Test acc.* (%)
Filter sizes	32-64-64-128	99.93	94.44	93.89
First dense layer	1024	99.36	94.21	93.81
Second dense layer	1024	98.39	96.64	96.15
Dropout	0.5	98.60	96.77	96.19
L2 Reg.	0.005	99.22	96.81	96.93
ES Patience	5	99.40	97.44	97.45
Data augmentation	–	93.08	91.35	90.23

Table 3 – Fine Tuning most important results obtained by the first and second approach. Added layers define the number of layers stacked on the cut net. Train, validation and test refer to the accuracy values for that part of the dataset.

Strategy	*Added Layers*	*Train (%)*	*Validation (%)*	*Test (%)*	Data Augm.
First	Only output	99.08	96.02	95.87	Yes
First	Only output	100.00	93.00	92.81	No
First	+256	99.42	94.87	94.99	Yes
First	+256	100.00	94.02	93.82	No
First	+256 with L_1 reg.	93.73	86.66	85.90	Yes
First	+256 with L_2 reg.	98.92	95.02	94.72	Yes
First	+512, +256	99.28	96.24	96.31	Yes
First	+512, +256 both with dropout	98.88	96.32	96.38	Yes
First	+1024, +512, +256 all with dropout	98.91	96.64	96.17	Yes
Second	Only output	99.62	96.99	96.92	Yes
Second	Only output	100.00	97.31	96.96	No
Second	+256	99.74	97.23	97.26	Yes
Second	+256	100.00	98.54	98.33	No
Second	+256 with L_2 reg.	99.74	97.71	97.68	Yes
Second	+256 with L_2 reg.	100	98.43	97.99	No
Second	+512	99.37	98.15	97.93	Yes
Second	+512	100.00	97.49	97.38	No
Second	+512 with L_2	99.76	97.83	98.02	Yes
Second	+512 with L_2	100.00	97.40	97.33	No