FRUIT CLASSIFIER

Deep Learning

Data Science and Advanced Analytics

Rita Franco	m20180081
Rodrigo Umbelino	m20180060
Vitor Manita	m20180054

01. Abstract

The aim of this project was to build a Neural Network model that was able to correctly classify fruit's photos into their respective categories. The data was collected from different sources and it is a combination of natural photos taken at supermarkets and not fruit icons in white background. In order to build the best classifier possible, we tried different architectures and different image pre-processing techniques. The models will be ranked by F1-score given that we are dealing with 16 unbalanced classes.

Data Source: *https://github.com/marcusklasson/GroceryStoreDataset*

02. Exploration

02.1. Data Description

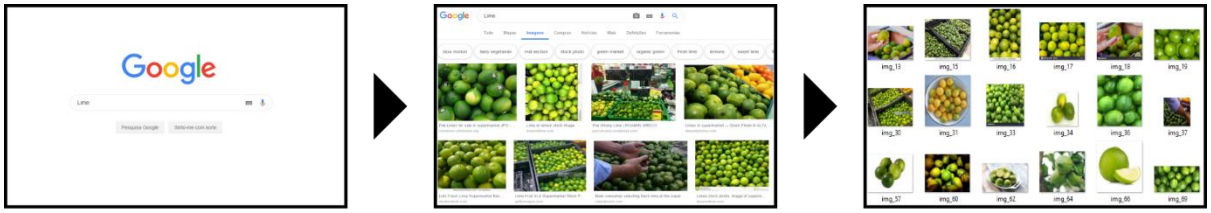
The original dataset is available on *Visual Data* and it is composed by over 5000 images divided in 81 classes of fruits, vegetables and carton packaged items but, for this project, we decided to only focus our attention of Fruits. The data consists in raw, natural pictures taken at supermarkets by smartphones. As shown in *Figure 1,* the photos are very different among each other, some are fruits held on people's hands and other are fruit stands. This was one of the main reasons why we chose this source of data instead of a dataset with a white background like one would find on *Kaggle*, given that real-world context photos impose a more challenging problem.

Figure 1. Sample Fruit images

During the project, we intend to build a neural network that correctly classifies the fruits provided according to their classes. Having an accurate model, the goal would be to integrate it with an application that aims to increase quality of life for visually impaired people, by helping them with day-to-day chores, like going to the grocery store to buy supplies. To do so, the user would simply point to the product/item with his/her smartphone's camera and the app would be able to correctly identify what type of grocery it is.

02.2. Increasing Dataset: Web Scrapping

The fruits dataset consists of a total of 19 different classes. Hence, our original data was reduced to 1117 images for testing and 1142 for training. Facing a small and imbalanced dataset, we decided to do something interesting in order to increase the number of training images: *Web Scraping*. Using Chrome Driver, we built a pipeline that increases the data with photos available on Google. *A priori*, we believe this will improve the generalization ability of out model, given that we are providing more examples and more different data to learn. A simplification of the process can be seen in *Figure 2*.



Chrome Driver searches for a specified term on Google

Then, it searches on Google Images and a parameter of how many images and pages to search is passed

Finally, it automatically selects each image in the search and stores it in the corresponding training fruit folder

Figure 2. Web Scrapping Process

After retrieving the images, we analyzed them manually to exclude the ones that did not fit our problem, such as images in white background, drawings or icons.

02.3. Final Dataset

After scraping Google images to gather more data, we decided to also reduce the original images stores for testing in half, in order to increase the training set size.

Having the data prepared, the next step would be to start testing different neural networks architectures and perfecting our model. By training some models we realized that two of our classes did not have enough images in order to be correctly interpreted by the models. Thus, our last modification to the dataset was to eliminate two classes: **satsumas** and **red-grapefruit**.

Finally, our dataset had 2270 images for training and 458 images for testing, divided across 16 classes as shown in *Figure 3.* The class with the biggest representation is *Apple* whereas the least represented fruits are *Plum, Papaya, Orange* and *Pineapple*.

Figure 3. Data Distribution per Class: Overview

Fruit	Total	Train		Test	
		Absolute Frq.	Relative Frq.	Absolute Frq.	Relative Frq.
Apple	544	414	74.73%	140	25.27%
Papaya	224	168	75.00%	56	25.00%
Banana	176	155	88.07%	21	11.93%
Melon	173	158	91.33%	15	8.67%
Kiwi	170	128	75.29%	42	24.71%
Pear	159	140	88.05%	19	11.95%
Mango	159	144	90.57%	15	9.43%
Lemon	150	127	84.67%	23	15.33%
Orange	141	123	87.23%	18	12.77%
Peach	139	122	87.33%	17	12.23%
Nectarine	131	104	79.39%	27	20.61%
Avocado	126	111	88.10%	15	11.90%
Lime	115	103	89.57%	12	10.43%
Plum	118	105	88.98%	13	11.02%
Passion-Fruit	104	93	89.42%	11	10.58%
Pineapple	89	75	84.27%	14	15.73%

Table 1. Data Distribution per Class: Detailed

03. Methodology

Having the data prepared, the first thing to do was to define a project strategy and decide whether to use an unbalanced dataset or to eliminate some images and balance it. On one hand, for this problem, the best would be to use a balanced dataset, so that all the classes have the same level of importance when training the model. On the other hand, like referred in the previous chapter, we don't have many images and dropping observations out would make us lose information that could be fed to the model and make it more accurate. Thus, we decided that the best way to go was to keep the size of the dataset and not risk losing information, rather than working with a balanced dataset. One additional thing we did to improve the generalization ability of the model was applying *Image Augmentation* in training data that, for each batch that is called while fitting, it generates artificial images of that batch with slight modifications, for example, rotated, cropped or zoomed.

It is important to highlight that, for each run of the models we experimented, we set aside 10% of the training data as validation and all metrics are show on this subset. The test set was used in the final metrics, when we want to evaluate the performance of our final model on unseen data.

03.1. Evaluating Models: F1-Score

We decided to start by implementing different neural networks architectures. All the networks will be fitted using cross validation and evaluate the models according to the *f1-score*, given that this metric has in consideration *precision* and *recall*, being more robust to the majority classes and is more sensitive to false positives, when comparing to accuracy. Since this metric is not available anymore in *keras*, we had to build a new scorer function.

$$f1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The following step will be to choose the best architecture and apply *Grid Search* on its parameters and obtain the best combination possible. This can either be implemented manually or using *Sklearn Grid Search CV*. Having the best architecture possible, we would like to apply different preprocessing methods in our images to realize which transformations make better results. Once again, these transformations will be evaluated by the *f1-score* over five folds of cross validation.

To safeguard all models, we add a *Callback* that saves the best model in each epoch and, additionally, we added an *Early Stop* feature that, if after 10 epochs nothing improves, it stops prematurely and starts the following run.

04. Exploring Architectures

After outlining a strategy, we began by exploring several network architectures. According to our knowledge obtained on the *Deep Learning* course, and after reading some literature, we knew the best option for this problem was to explore the *Convolutional Neural Network* structure, since it sort of performs a feature extraction from the image elements, being capable of learning patterns from the image such as shadows and edges. We tried some dummy trial and error *CNN* architectures of our own, presented on *Table 2,* and searched for existing available architecture solutions.

CNN_A	CNN_B	CNN_C
Conv2D (relu)	Conv2D (relu)	Conv2D (relu)
Maxpooling	Maxpooling	Maxpooling
Conv2D (relu)	Conv2D (relu)	Batch Normalization
Maxpooling	Maxpooling	Conv2D (relu)
Flatten	Flatten	Maxpooling
Dense (relu)	Droupout (0.5)	Flatten
Dense (softmax)	Dense (relu)	Dense (relu)
	Dense (softmax)	Dense (softmax)

Table 2. Initial built Convolution Neural Networks

After some readings, we found 4 other architectures that look promising and worth exploring, they are: *LeNet 5, AlexNet, VGG16* and *Resnet (18, 50, 101 & 152)*. A brief description of the models can be found on *Table 3*.

LeNet5	AlexNet	VGG16	Resnet
Conv2D (relu)	Conv2D (relu)	Block	Conv2_x
Maxpooling	Maxpooling	Maxpooling	Conv3_x
Conv2D (relu)	Conv2D (relu)	Block	Conv4_x
Maxpooling	Maxpooling	Maxpooling	Conv5_x
Flatten	Conv2D (relu)	Block	Dense(softmax)
Dense (relu)	Conv2D (relu)	Maxpooling	
Dense (softmax)	Maxpooling	Block	
	Flatten	Maxpooling	
	Dense (relu)	Block	
	Droupout (0.5)	Maxpooling	
	Dense (relu)	GlobalAvgPooling	
	Dropout (0.5)	Dense (relu)	
	Dense (softmax)	Dense (relu)	
		Dense (softmax)	

Table 3. Architectures used as Reference

04.1. Initial Benchmark

Having these architectures set, we ran the algorithms on 5 different seeds and 25 epochs on each seed and evaluated the generalization ability on the validation subset. What we noticed initially was that the architectures found online, given that they are far more complex, required a more intense computational effort. From *Figure 4* we can see that, overall, using bigger images sizes of 64px does not improve a lot the results of the models, thus, our first conclusion is to use all images with 50px for future benchmarks and analysis. The second observation is that the *CNN* architecture experimented by the group appear to present better results that the other models used as reference, namely, *CNN_B* presents a good initial *f1-score* around 60% and low variance on the results of the 5 runs. Model *Resnet18* also looks promising, however, due to the run times, a simpler model such as *CNN_B* is preferable.

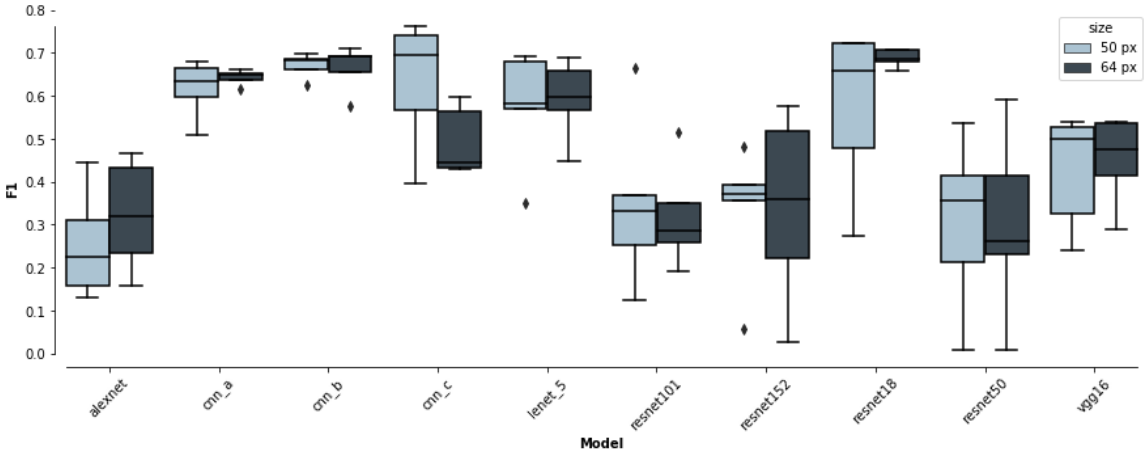


Figure 4. Initial Architectures benchmark

05. Best Architecture fine tuning

After performing the initial benchmarking, we decided that it would be important to find the set of parameters that would optimize the *CNN_02* architecture, the one that appeared more promising. To achieve this, *Grid Search* was the obvious technique to be applied in this situation. However, we were not able to run the Grid Search from the *Sklearn* package in our model, since it was crashing on our devices. To overcome this, we implemented the Grid Search concept from scratch, using nested loops while comparing the different values of the Grid Search Space. The evaluated model's parameters and respective search spaces are described in *Table 4.*

Parameter	Definition	Grid Search Space
neurons	The number of neurons in the layer	[50,70,80,100,120]
drop_out	Fraction of input units that are dropped out at each update during the training time	[0.1, 0.3, 0.5,0.8, 0.9]
optimizer	Optimization algorithm used in the model	['adam', 'sgd']
pooling	Type of method to use in the Pooling layer	[AveragePooling, MaxPooling]

Table 4. Grid Search Parameters

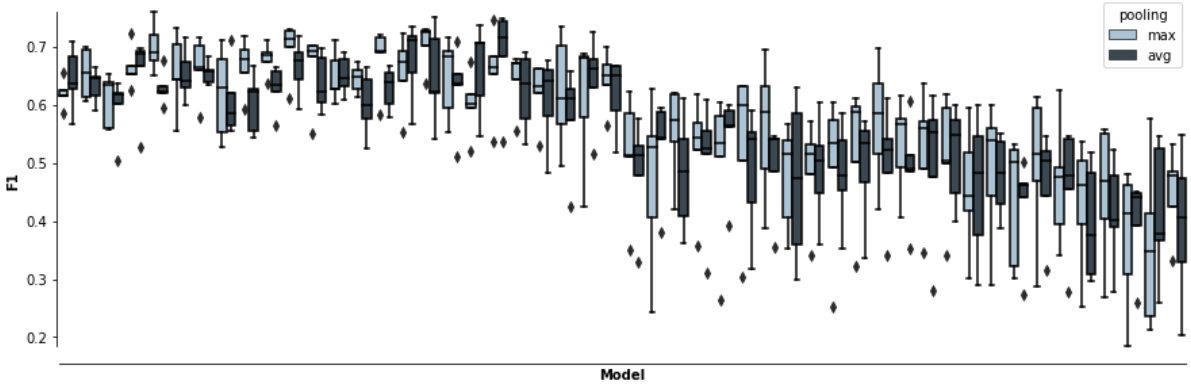


Figure 5. Manual Grid Search Overall

By analyzing the results of *Figure 5*, we can conclude that models with *MaxPooling* tend to exhibit the best average *F1-Score*, when comparing to models that have *AveragePooling*. If we take into consideration only the models that present a *F1-Score* greater than 0.5, we can also see that *MaxPooling* has, for the most part, less variability and dispersion in results than *AveragePooling*. If we look at the lower end, it is observable that worst-performing models not only present lower *F1-Scores*, but they are also much less consistent and more disperse.

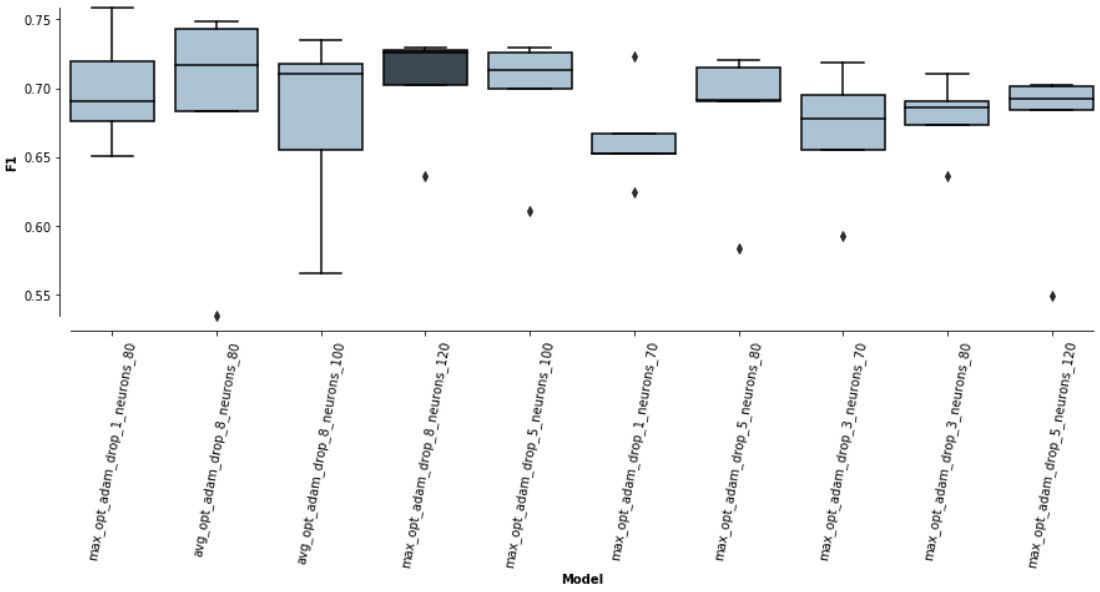


Figure 6. Top 10 Grid Search combinations

Taking into consideration results presented in *Figure 6*, we can see that the implemented models seem to greatly benefit from the usage of the Adam optimizer, since it clearly outperforms Stochastic Gradient Descent. In addition to this, models that have higher dropout rates seem to have better results on average, when comparing to ones that do not have such penalizing fractions. Regarding the number of neurons, there is not a particularly distinct pattern to be deducted from the results, besides the fact that a smaller number of neurons tends to worsen the models' performance, on average.

Tunned Model
The final optimized parameters for our Convolutional Neural Network consist in 120 neurons, 0.8 dropout rate, Adam optimizer, and the MaxPooling method for the Pooling layer.

06. Filters

Having the results for our best model, we decided to explore different image processing techniques. More specifically, we thought that it would be useful to experiment several types of image transformation filters, which aim to extract the most important features from the image data. Having said that, the 4 implemented methods are described in *Table 5* and its corresponding effect can be briefly seen on *Figure 7*.

Filter	Description
Sharp	Aims to sharpen the data to deblur the images and make them sharper.
Enhance	A combination of different filters that apply colour correction and edge smoothing, to increase the overall quality of the image.
Shade	A filter that increases the weights of the image's shadows.
Contours	Converts the image to greyscale, making only the outlines visible. It extracts the shape of the objects rather than focusing on the colour itself.

Table 5. Image Filters Description

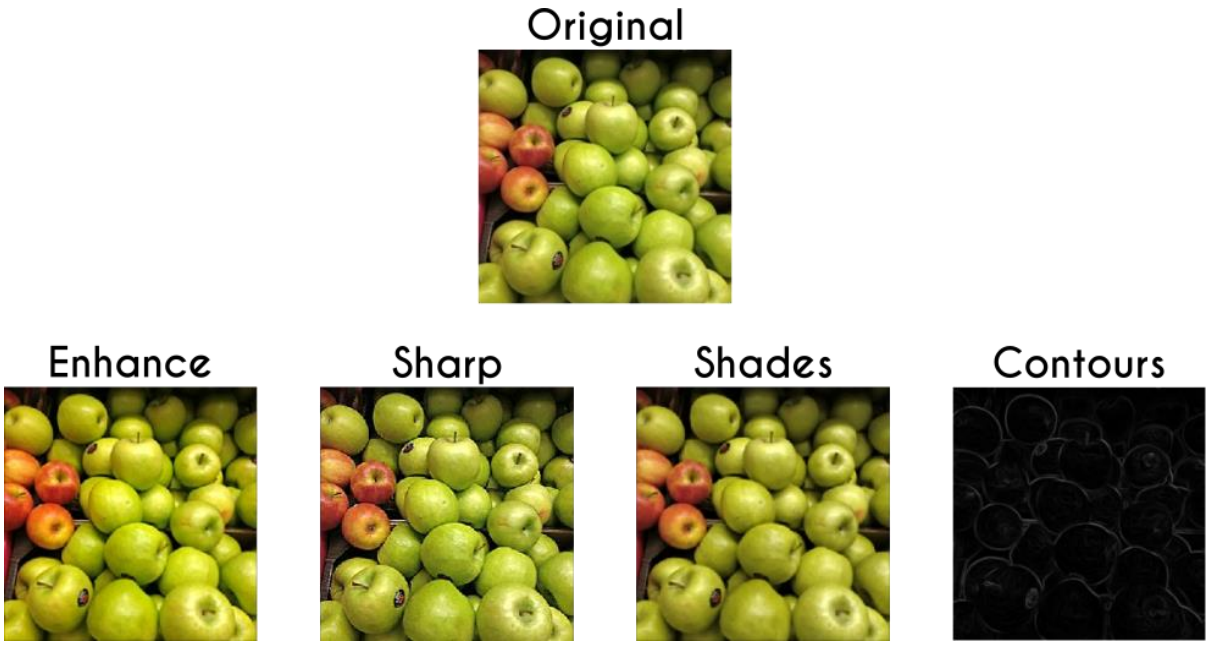


Figure 7. Image Filters

06.1. Filters Benchmark

After implementing the filters that are shown in *Figure 7*, it was time to benchmark them. We did so by running each filter on 5 different seeds, with 25 epochs per each. This way, we could evaluate the model's generalization ability with each filter enabled and compare the results with the original image. The results of these benchmarks are described in *Figure 8*.

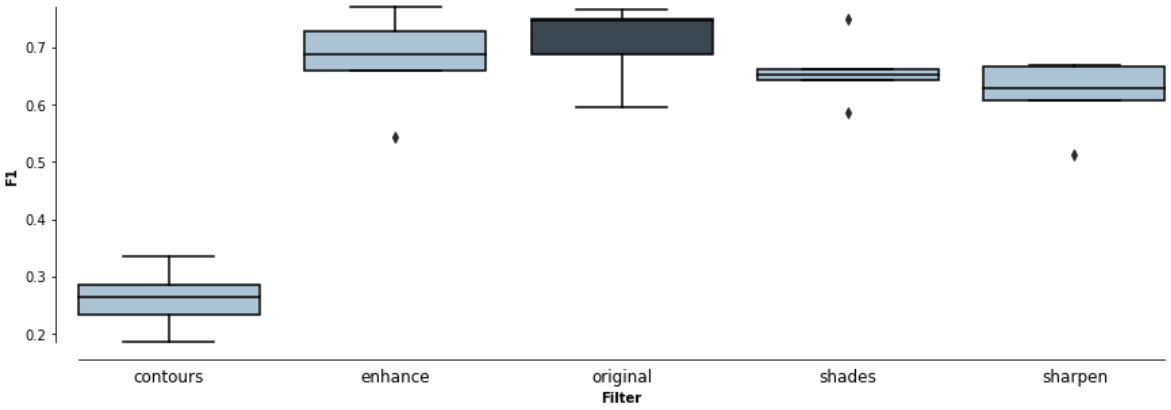


Figure 8. Filter Benchmarks

As we can infer from the results presented in *Figure 8*, the original images (the ones with no filter applied), were the ones that performed the best in our model. This goes against our *a priori* assumptions that the Enhance Filter would outperform the original one. This might have happened because some of the colour corrections applied by this filter can be too drastic in some of our dataset's images. In addition to this, the poor performance of the Contours Filter comes as no surprise, because despite effectively extracting the contours and edges, we lose the hue/colour feature from each image, which is a very important feature giving the nature of our problem. Lastly, we were also surprised by the relatively good performance and low variance of the Shades Filter, which we assumed that would perform worse, since in some images, the weights of the shadows imposed by this method can drastically change the data, when compared to the original one.

07. Final Runs and Access

Having everything set in terms of image preprocessing and a final architecture picked, we decided that, to have a better trained model and better statistics, instead of the usual 25 runs, we would run our best model with 200 epochs with a validation split of 10% used for accessing the quality of the model across all generations. To ensure we don't overfit the model, we tracked the progress at each iteration and implemented the *callback* that saves the best model using the validation loss as metric to compare the best solution so far.

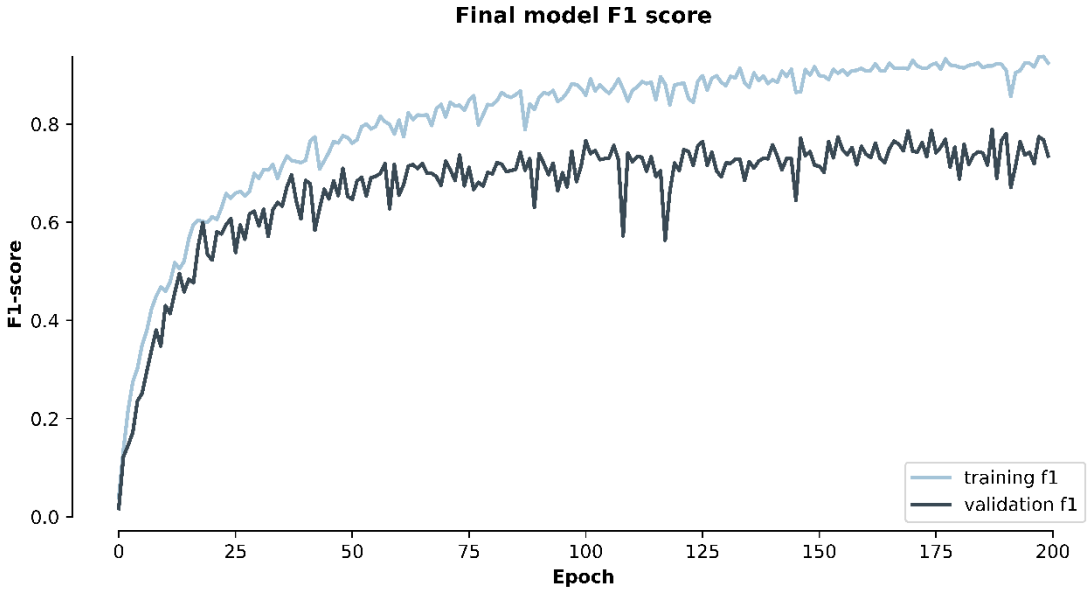


Figure 9. Final Model F1-Score

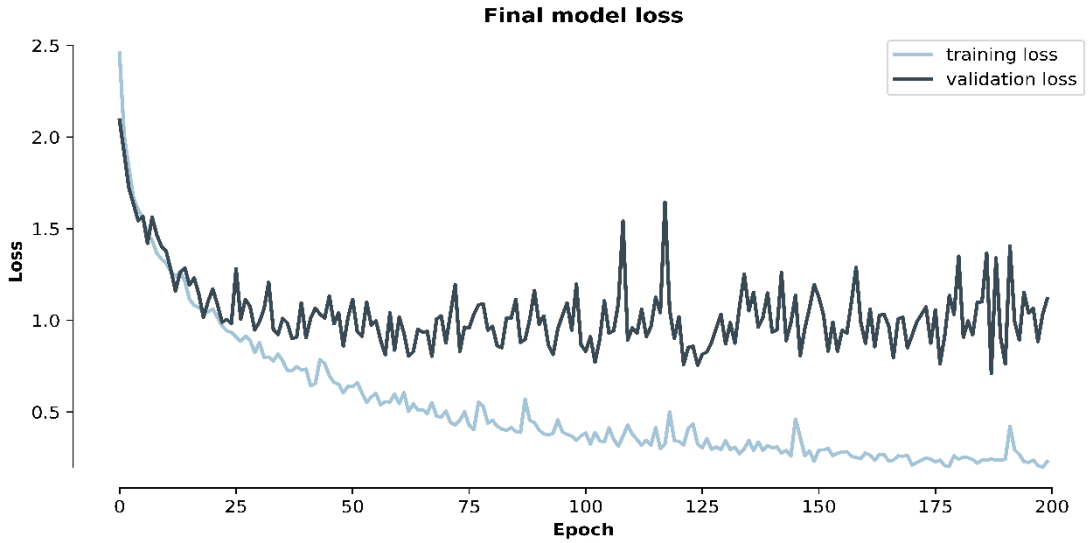


Figure 10. Final Model Loss

By analysing the plots, even though the *callback* implemented stored the best model, we can observe that, in *Figure 9*, a good cut-off point would be around 65 epochs. In *Figure 10,* we can support this decision by searching a point where the loss ceases to decrease and stagnates, that would be around the same number of generations, from 60 to 70 generations. The ideal stop point would be in this neighbourhood, to have a good generalization ability and not overfit the model.

07.1. Access

Lastly, since the beginning of the project that we had a subset of untouched and unseen images with the sole objective of being used as test and to access the generalization ability of our model. The results for *Recall*, *Precision* and *F1* score can be observed on *Table 6*.

Fruit	Precision	Recall	F1	Support
Apple	0.92	0.84	0.88	140
Avocado	0.70	0.84	0.76	19
Banana	0.62	0.87	0.72	15
Kiwi	0.90	0.78	0.84	23
Lemon	0.83	0.90	0.86	21
Lime	0.67	0.80	0.73	15
Mango	0.35	0.47	0.40	15
Melon	0.84	0.86	0.85	42
Nectarine	0.55	0.94	0.70	17
Orange	1.00	0.89	0.94	27
Papaya	0.69	0.92	0.79	12
Passion-fruit	0.69	0.64	0.67	14
Peach	0.71	0.67	0.69	18
Pear	0.86	0.66	0.75	56
Pineapple	0.88	0.54	0.67	13
Plum	0.83	0.91	0.87	11
Micro avg	0.80	0.80	0.80	458
Macro avg	0.75	0.78	0.76	458
Weighted avg	0.82	0.80	0.80	458

Table 6. Metrics on test subset

Overall, we are satisfied with the result, given that we are dealing with real-world photos taken at a supermarket and have a *weighted F1-score* on unseen data of 80%. At a closer and smaller granularity, we can see that the final convolutional network is pretty good at detecting *Apples*, *Oranges*, *Plums*, *Lemons*, *Melons* and *Kiwis*, providing *F1-scores* above the 80% average threshold. On the other hand, *Mango* appears to be a hard fruit to classify for this model.

08. Conclusions

Concluding, this was a very interesting project and subject to approach. It was also challenging, given that we were dealing with real-world photos taken at supermarkets and images retrieved from Google images. The computational effort necessary to run all models was also an obstacle that stopped us from doing a deeper benchmark and fine tuning.

However, we did expect a simple model such as the *CNN* we build would outperform the existing models of *AlexNet*, *ResNet*, *LeNet5* and *VGG16* and that was indeed what happen. Then, having our final model, we explored a manual search for the best parameters. Lastly, for the preprocess part, we tried different image filters, hoping that this would improve the generalization ability of the model and, what happened in the end was that it did not, in fact, improve the *F1-Score* of the model tested on a validation subset.

Having everything ready, we ran the final model with 200 epochs to have a better robust algorithm and applied it to the testing subset, what resulted in an average *F1-Score* of 80%, revealing a good generalization ability.

Due to time restrictions, we were not able to implement a technique called *Progressive Resizing* [1], that we believed it could really improve the generalization ability of the model, since it uses small sized images to learn basic features and applies/transfers that knowledge into training bigger images.

Lastly, we believed that, even this project only deals with fruit images, it gave us a grasp of how *Deep Learning* research can offer solutions to improve the life of visual impaired people and, from fruits, it is possible to build so much more.

09. References

[1] A. Bilogur, "Boost your CNN image classifier performance with progressive resizing in Keras," 2 april 2019. [Online]. Available: https://towardsdatascience.com/boost-your-cnn-image-classifier-performance-with-progressive-resizing-in-keras-a7d96da06e20. [Accessed on May 2019].

[2] P. Rodriguez, "Deep Learning, Applied. Project #1," 22 January 2017. [Online]. Available: https://blog.stratospark.com/deep-learning-applied-food-classification-deep-learning-keras.html#Image-Augmentation. Accessed on May 2019].

[3] E. Allibhai, "Building a Convolutional Neural Network (CNN) in Keras," 17 October 2018. [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5. [Accessed on May 2019].

[4] D. Pradilla, "Classifying fruits with a Convolutional Neural Network in Keras," [Online]. Available: https://www.danielpradilla.info/blog/classifying-convolutional-neural-network-keras/. [Accessed on May 2019].

[5] S. W. G. J. P. P. Yudong Zhang, "Fruit classification using computer vision and feedforward neural network," Journal of Food Engineering, vol. 143, pp. 167-177, 2014.

[6] B. Abdelkader, "Keras ….4 types of Fruits …classification Part II," 13 March 2018. [Online]. Available: https://medium.com/@bouhafsabdelkader/keras-4-types-of-fruits-classification-part-ii-2a007d16c002. [Accessed on May 2019].