Machine Learning - Homework 5

João Cardoso
81361

Mafalda Ferreira
81613

Mário Cardoso
83523

Group 3 - Alameda

https://colab.research.google.com/drive/14DELdGHKLYr3YFIXGIiLY2v91jb9BO3N#scrollTo=_CL4YWC-WU6z

This program assumes there is a folder inside the **content**/ folder named **fruits-360** with two folders inside: **Training** and **Test**. In both folders, it is assumed that all the apples are inside the **Apple** folder and the same for the Pears, Lemons and Oranges.

In is possible to upload folders to a colab notebook by zipping the folder, upload the .zip and then unzip it in the colab notebook by running the snippet: `!unzip fruits-360.zip`.

1 Introduction

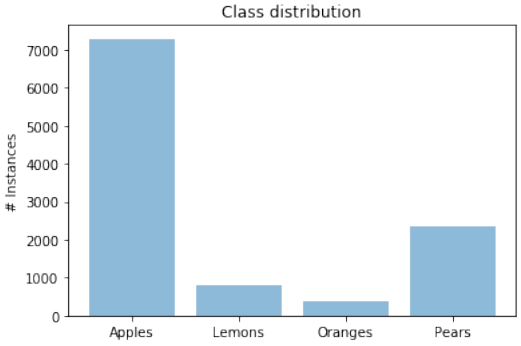
The goal of this homework was to develop a neural network able to identify objects in an image. Particularly, our network should be able to answer whether a fruit is present in an image or not and to identify which fruit it is. The classes that will be recognizable are apples, oranges, lemons and pears. There are several approaches to solve this problem, such as creating a convolutional neural network from scratch or downloading an already trained neural network and either using it as it is or retraining parts of it, known as transfer learning. Due to time constraints we ended up foregoing the possibility of creating a network from the ground up. As such we downloaded several pre-trained neural networks, retrained some layers and compared the results from each one to choose the best one.

1.1 Choosing the neural network

As was said before in order to find the best network we chose two and then compared their results. The chosen networks were MobileNet and VGG16. After running several tests we found that both had similar accuracy. However, MobileNet is 32 times smaller than VGG16 and as such was able to achieve similar results to the latter in 10 times less the amount of time. As a result we ended up choosing the MobileNet Network to use and retrain.

1.2 Splitting the dataset

The dataset used for this exercise was the fruits dataset provided by Kaggle. Due to the requirements of the exercise, the images regarding fruits not belonging to either Apples, Lemons, Pears and Oranges were removed as well as the images with several fruits. All the images belonging to different variations of the required classes (Apples, Lemons, Oranges and Pears) were grouped into the same folder. To split the dataset, we utilized ImageDataGenerator which is an image preprocessing package from keras. For the train-validation split we decided to use an 90-10 split. For the mobile net neural network, it was necessary to apply the mobile net preprocessing input to the ImageDataGenerator. To obtain the batches of images we utilized the flow_from_directory method which takes a path and returns the images within that path given the specified parameters. For the validation and training batches the training directory from the dataset was provided. For the test batch the test directory from the dataset was provided. To label these images, during the flow_from_directory step, we specified an array with the corresponding classes for each folder. In order to better understand the distribution of the classes, after splitting we plotted the number of instances belonging to each class for the training dataset directory:



Analyzing this bar chart we can notice right away that the dataset is unbalanced since most instances belong to the class Apple and very few instances belong to the classes Lemon and Orange.

1.3 Training process

As mentioned before, we utilized the transfer learning method to obtain the neural network which consists on using a neural network that has been trained with a large dataset similar to the one we have and retrain parts of it to adapt to the current problem.

The developed models were compiled utilizing Adam optimizer, with a learning rate of 0.001 and utilizing categorical crossentropy as the loss function. This value of the learning rate was chosen to be small in order to not drastically affect the already trained MobileNet network weights, which are already quite good.

To fit our model with the training and validation set a group of parameters needed to be specified, namely batch sizes for the training and validation sets as well as steps per epoch and number of epochs. Utilizing GPU accelerated hardware, the batch size selected was 64 for all the sets and the steps per epoch selected was images in that set / batch size. Utilizing the 90-10 split, for the training set the steps per epoch were 152 and for the validation set the steps per epoch were 17. These particular values regarding batch size and steps per epoch were selected in order to utilize every training image available in one epoch. The following sections all consider a batch size of 64, steps per epoch of images in set / batch size, number of epochs of 30 and keras utilizing tensorflow-gpu backend.

Having chosen the MobileNet network we proceeded onto training different numbers of layers to understand how this changed the algorithm's performance. First we created a layer with 4 units (since we want to classify 4 classes) and softmax activation function that received as input the output of the base MobileNet network. Afterwards we created a model that received as input the same input as the MobileNet network and outputted the results of the previously mentioned layer. We froze every layer except the newly created one in order to avoid retraining weights. This was done to serve as a baseline to improve on and compare with future models. The second approach taken was to retrain a certain number of final layers from the MobileNet network. To accomplish this, we attempted to retrain the last 5 layers. The previously mentioned layer took as input the output of the 6th to last MobileNet layer. The model was then created utilizing the same methodology as above except the last 5 layers were not frozen to training. This retraining provided some more appealing results than the previous approach. The previous approach provided some good results but we tried to improve upon it by applying different regularizers to the created layer as well as varying the number of trained layers. The following tables displays the results for the model without retraining, the model with retraining of the last 5 layers and the model with retraining of the last 5 layers with L2(Ridge) regularization with term 0.01 respectively.

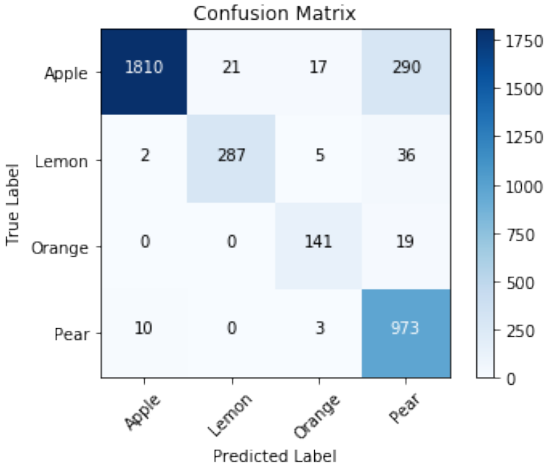
Training Accuracy	Training Loss
89.05%	0.3897
100%	6.9786e-04
100%	0.0017

Validation Accuracy	Validation Loss
92.95%	0.2295
97.77%	0.1273
70.32%	0.6169

Given these results, we chose the model with retraining of the last 5 layers without regularization to proceed into the self-explanatory testing phase.

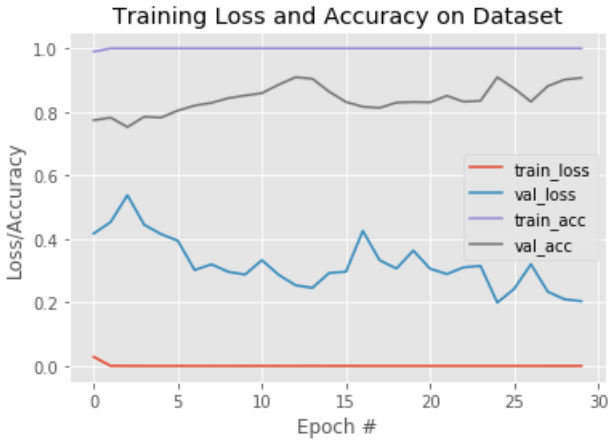
1.3.1 Testing process

After fitting the model with training and validation images, we proceeded to test how the model behaved when faced with unseen instances. To do this, we utilized the reserved test batches, predicted their labels using the model and compared the predicted labels with the true labels. We plotted these results with a confusion matrix, with the true labels on the y axis and the predicted labels on the x axis. The diagonal cells represent correct predictions, the remaining cells represent incorrect predictions.



Besides the confusion matrix, we also calculated the cumulative loss (categorical crossentropy) and accuracy for a given test batch. For the batches used in the confusion matrix above, the following results were obtained:

$$Loss : 0.0501 \qquad Accuracy : 97.39\%$$



1.3.2 Regularization

Regularization is used to dampen the effect of variance and to do so without inflicting too much of a change in bias. It achieves this by adding a shrinkage quantity to the loss function that also needs to be minimized. This will punish coefficients with high values and in Lasso regularization will also be able to set coefficients to zero, effectively working as a feature selector. After running several models with different values for both regularization types we found that both regressions had similar performance, however they were not able to improve on the regular model. As a result we did not include regularization in our final model.