REPORT MACHINE LEARNING HOMEWORK 2

IMAGE CLASSIFICATION

“Classification of Images through the usage of Neural Networks”

George Ciprian Telinoiu - 1796513

23/12/2020

ABSTRACT

The task that we have been assigned was to analyze a custom dataset generated randomly from our student id. We were assigned 8 different classes all containing different images, gathered from the Bing Search Engine that had to be classified by our model.

Our job was to define and implement any kind of Machine Learning to deal with the task. Given the fact that we were working with images, considering also the subject covered during the second part of the course I decided to approach this task using Neural Network, specifically Convolutional ones.

Moreover those were the 8 classes assigned to me:

- *Blueberries,*
- *Energy Drink,*
- *Honey,*
- *Latex Gloves,*
- *Vegetable Chips & Crisps,*
- *Accent Plate,*
- *Grocery Bag,*
- *Plastic Knife.*

0. PRELIMINARIES

The following report will be divided into several subsections, following the order in which I implemented and reasoned around the models presented for the given dataset, specifically:

1. Pre-Processing & Dataset Analysis
2. Model Selection
3. Training & Training Results Analysis
4. Classification & Classification Results Analysis
5. Conclusions

All the implementation was developed within the *Google Colaboratory* environment, thoroughly making use of the keras library, utilizing tensorflow to deal with training and image processing.

1. Pre-Processing & Dataset Analysis

The Dataset has been firstly un-archived within my PC and then uploaded to Google Drive in order to be able to work with it within Google Colab.
I have created a primary directory called "Dataset" and within it each class had its separate folder with all images attached to it. The name of each folder was exactly the name of the class, this was crucial in order to later make use of this split.

The Dataset was composed of 9124 RGB images, whose original dimension was of 256x256. Specifically I had, in order of dimension of images:

1. Honey **1297**
2. Energy Drink **1232**
3. Vegetable Chips & Crisps **1200**
4. Blueberries **1101**
5. Grocery Bag **1098**
6. Accent Plate **1073**
7. Plastic Knife **1064**
8. Latex Gloves **1059**

No test or validation set were provided.

Before starting to actually work on the dataset I wanted to analyze it a bit better in order to understand what kind of images I had and to have an idea, besides basing my hypothesis on the number of images, of the classes that might end up having some problems and the ones that might return the best results.

We are also not considering the possibility of also having Binary Classification for each class, for example in the folder "*Plastic Knife*" even if we find an actual Knife we are not considering it as a misclassification.

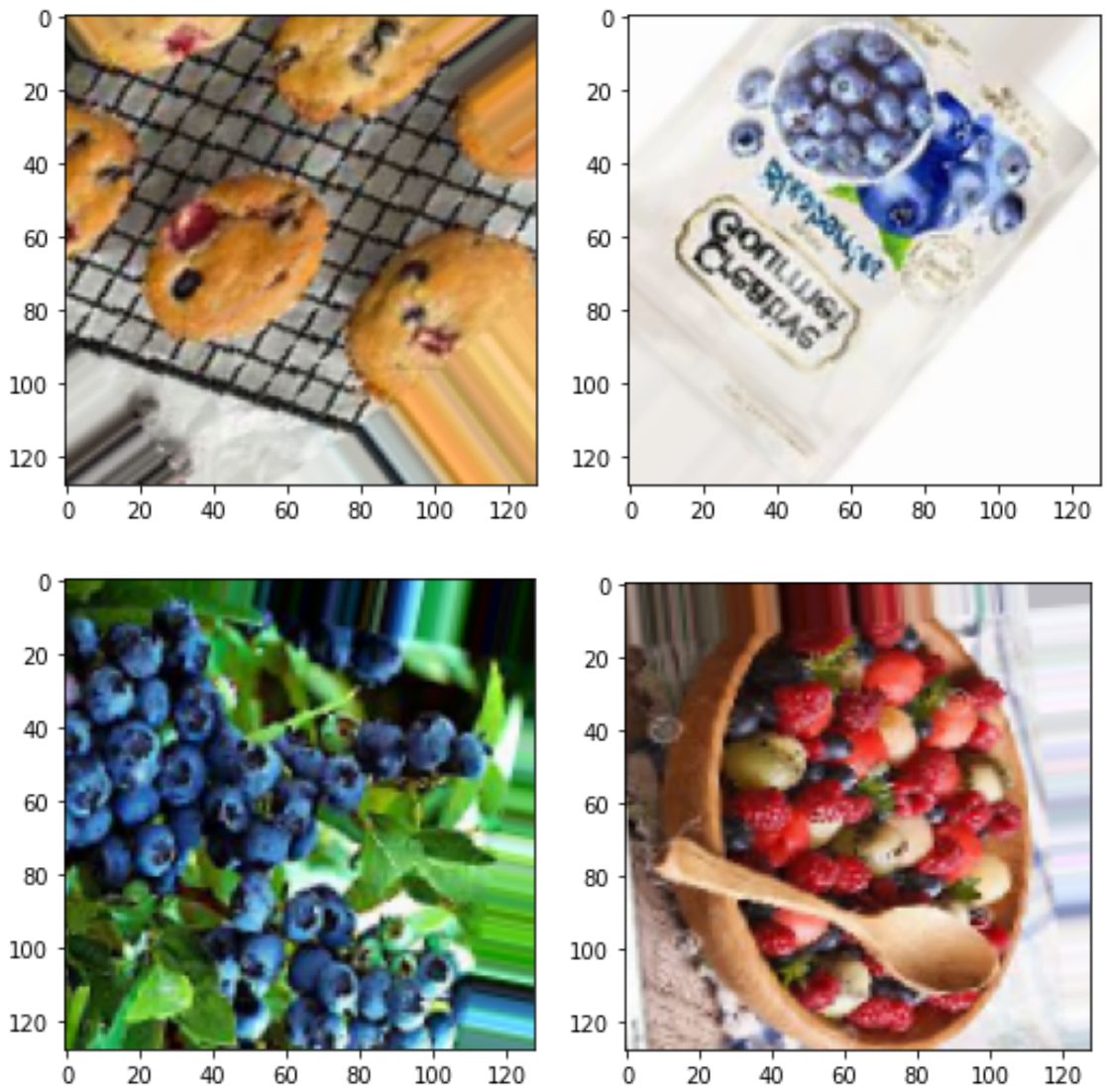
The most critical ones have to be for sure *"Honey"* & *"Grocery Bag"*, considering how hard it is to determine in certain images what exactly should be considered as the reference object. I decided, after reaching out also to the professor, to leave all images as they are and not apply any kind of pre-processing to those classes; keeping the ambiguous images as noise for the dataset.
On the other hand, classes such as "*Plastic Knife" & "Accent Plate*" are unambiguously identifiable, therefore I expected them to perform best and show the least cases of FP or FN.

Having now analyzed the Dataset, I proceeded to prepare it for the Training of the several Neural Networks. What we needed was to create several batches of tensor images that the Network will use for it's training.
I have decided to utilise the "*tf.keras.preprocessing.image.ImageDataGenerator*"

function to prepare the batches of images and then use the *"flow_from_directory"* function on it to get the images from Google Drive. At this point I had to decide if I wanted utilise any kind of Data Augmentation on the Dataset, which later turned out to be crucial to get good results. Firstly I decided only to apply a simple rescale, keeping the images as they were, to have a base point for further models.
On other models on the other hand I have applied several data augmentation techniques to broaden the dataset and avoid overfitting, that indeed happened only when I tried to run without data augmentation. Here's a list of the DA used:

- *rotation_range* : **90**
- *width_shift_range* : **[-10,10]**
- *height_shift_range* : **[-10,10]**
- *horizontal_flip* : **True**
- *vertical_flip* : **True**

Here's a few examples of the result of the aforementioned data augmentation:



At this point I also had to decide the split that I wanted to perform on the Dataset to extract some sort of test_set from it. I decided to stick to a standard split, going for 20% of the total Dataset as Test Set. I also decided to keep the shuffle at *false* to be able to keep a certain level of costancy between various models.

The final parameters that I had to handle were dimensionality of images and batch size, whose standard values were respectively 256x256 and 32.
Considering that the Dataset was large enough and our first interest wasn't to get the absolute best result out of it, I decided for the majority of tests to keep a dimensionality of 128x128 and a batch size of 64.

At this point we are ready to approach Model Selection and Training.

2. Model Selection

When trying to select the Models to evaluate my considerations focused on it's dimensions, speed of training, eventual accuracy and customizability. The ones that I picked were: VGG16, AlexNet & LeNet. They all represent Convolutional Neural Networks, which as mentioned in the Premise are the best alternative when working with image classification. I considered that they were all different enough and could give me the chance to analyse their differences, while looking to get a good result on my task. Here's a summary of the structure of each model (*see next page)*.

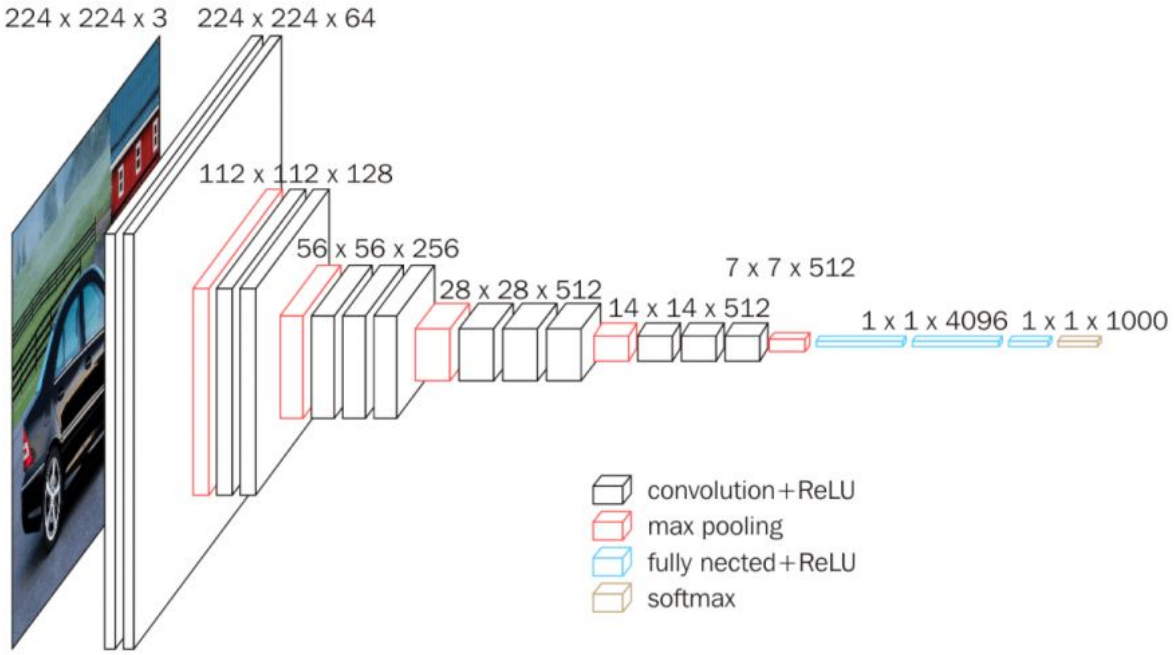
For VGG16 an approach of transfer learning was applied, specifically by freezing the majority of the weights and adding the last layers to perform classification for our classes. Besides this approach there was also the one of performing fine tuning, but I considered it not to be necessary or to be considered for this kind of task.

AlexNet had to be trained from scratch, and wanted to represent a very deep and complex Convolutional Neural Network. On the other hand I wanted to see how a much simpler Network as LeNet would perform on this dataset.

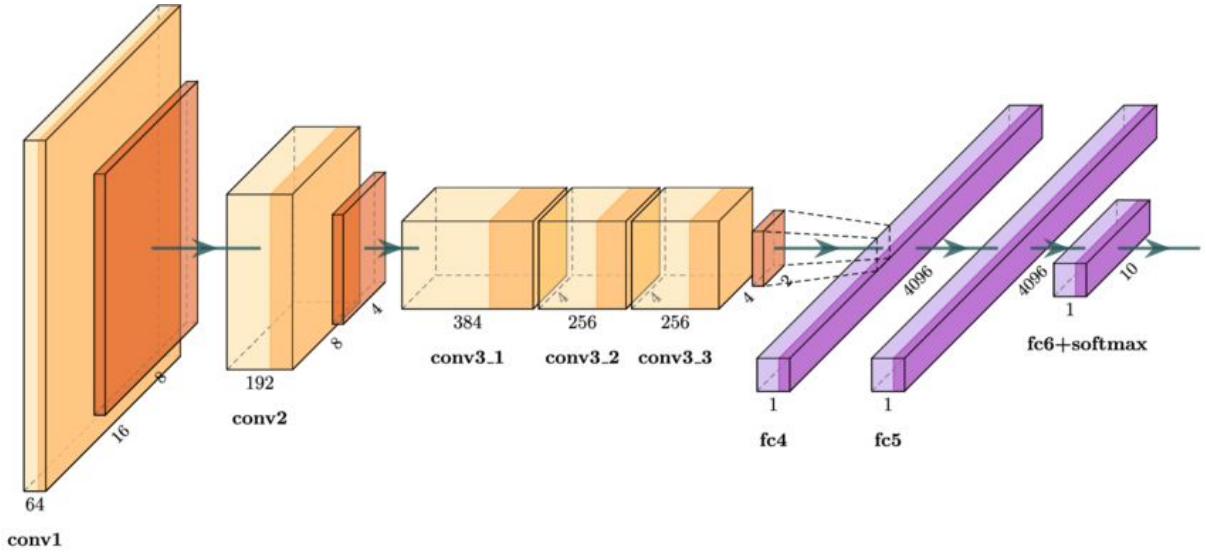
Considering our Multi-Class problem all Networks were compiled with loss = *"categorical_crossentropy"* and for metrics I decided to use *"accuracy"*. The optimizers that I've tried on the other hand are both *"adam"* and *"SGD"*

Finally I decided not to modify the Learning Rate, since early tests led to high instability and divergence.

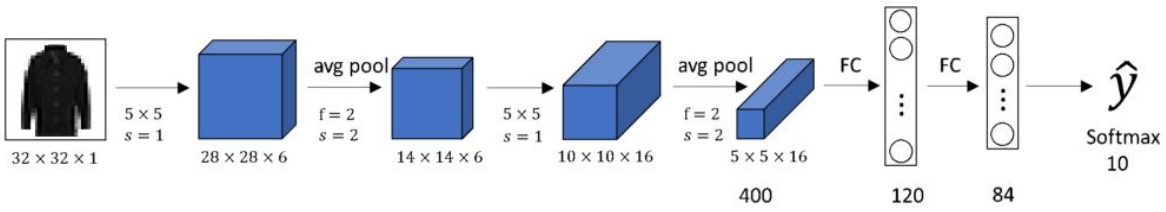
VGG16



AlexNet



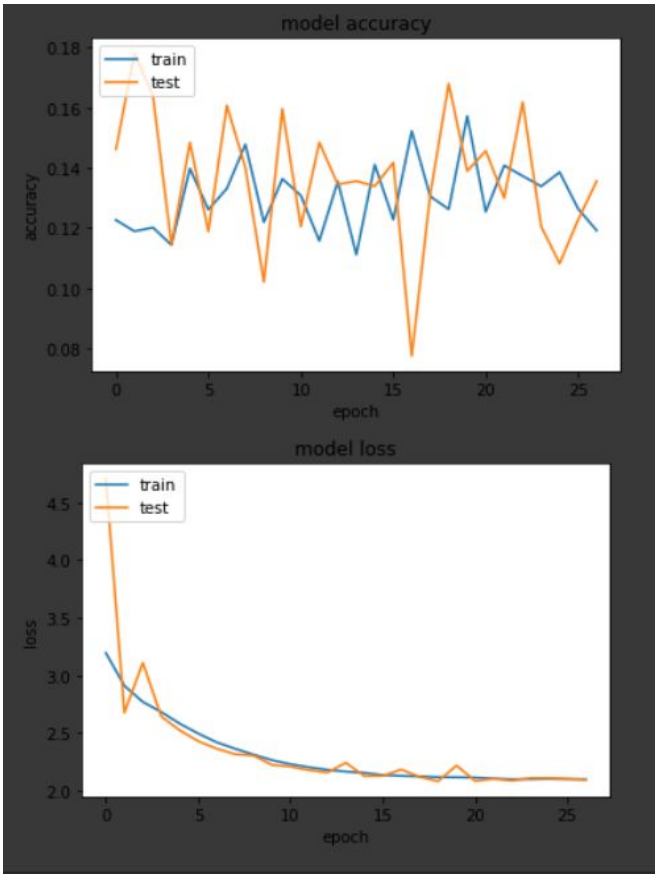
LeNet



3. Training & Training Result Analysis

I will now analyse the Training of each Model, commenting its evolution and the modifications that I have adopted while training.

3.1. AlexNet



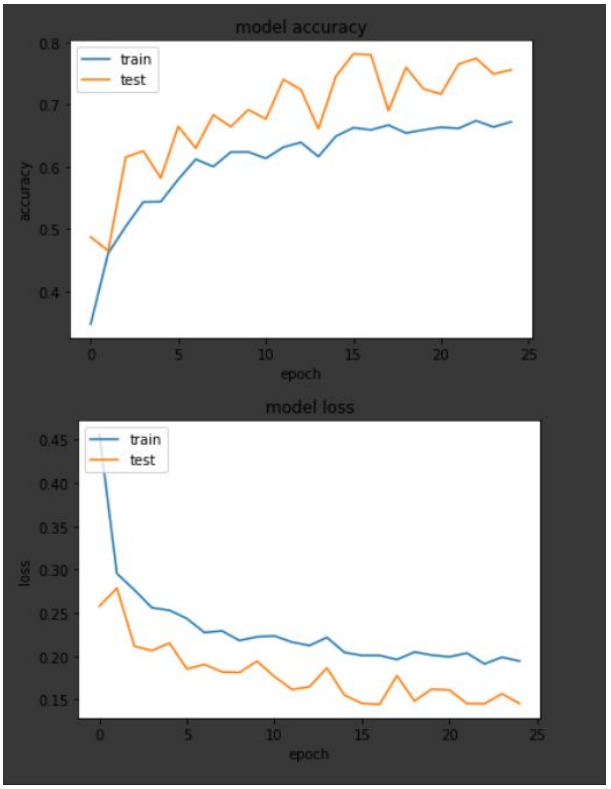
I wanted to start with the Network that I imagined would have taken most time, but eventually I ended up switching my interest to the other 2 networks considering the very bad results that I have obtained in my tests. First of all it presented the highest starting Loss at almost 5. Fortunately this drastically improved as we can see in the graph, plateauing at a bit above 2. On the other hand the accuracy never managed to reach a value superior to 0.2, neither for the training or test set. Those results were returned by a halt of the fit function considering that I had included a policy of Early stopping. I have also tried pushing further with higher epochs but as expected the situation did not improve. Finally neither changing the optimizer to SDG made any difference. Considering those results I haven't proceeded to analysing the results and therefore perform any kind of prediction for this model. This was also due to the much more interesting results obtained from training LeNet and VGG16. Surely there is a way to improve this situation, maybe from increasing the LR.

3.2. VGG16

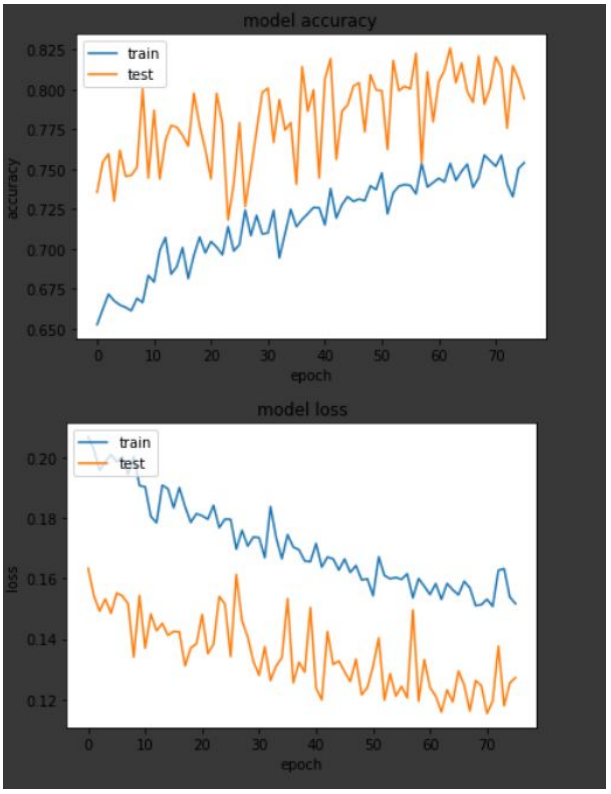
With this training, as expected, results were much more promising, in fact they led to the best model that I have managed to train. Before getting there I will show the various results that I got.

For this model I also decided to apply a policy of Early Stopping at the beginning in order to better understand what kind of results I was able to reach with a restricted number of epochs. In the following page I present the graphs of the various training that I have tried and that I will now comment.

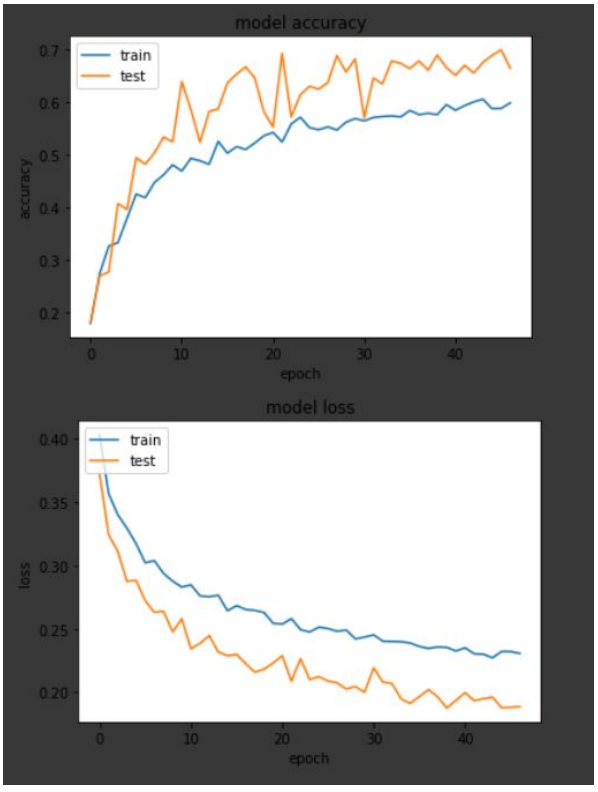
VGG16 - 128X128 - ADAM - 25 EPOCH



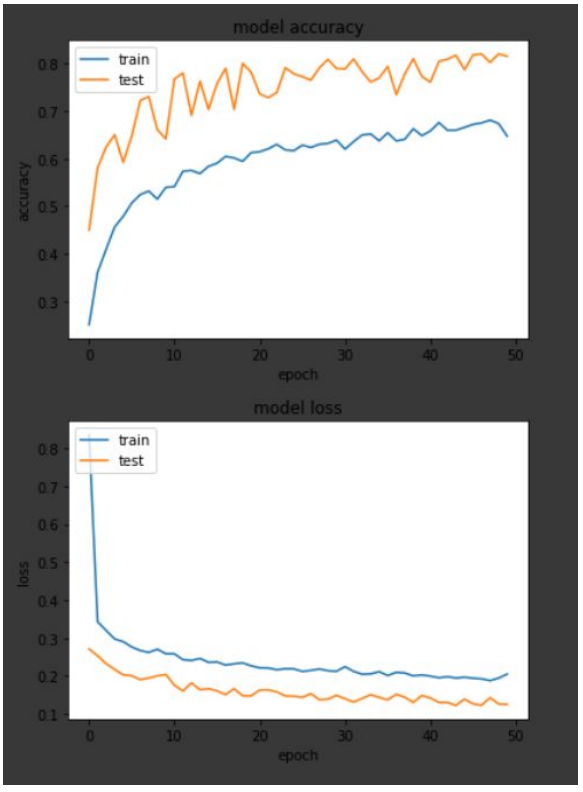
VGG16 - 128X128 - ADAM - 100 EPOCH



VGG16 - 128X128 - SDG - 50 EPOCH



VGG16 - 256X256 - ADAM - 50 EPOCH



The first training had no problems, with both train and test set improving with each epoch with no major hiccups. Being curious of how this training would have continued, I decided to keep training until epoch 100, and indeed even though we gained a bit of performance, the trade-off is definitely not worth it.

Having those results as a base case, I wanted to see how much of a difference changing the optimizer could bring, therefore I compiled a new model setting "SGD" as a classifier. Being it a more complex network I expected Adam to perform better, but at the same time the difference wasn't that big. It's very interesting though to see how with SDG the curve is much more unstable, especially when computing accuracy.

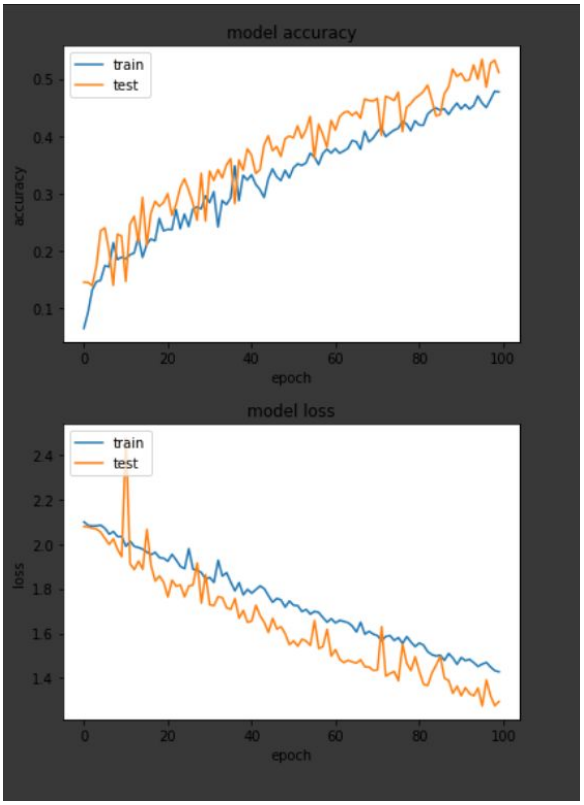
Finally, considering the knowledge gathered and the results that I will discuss in the next section, I wanted to improve the model by providing it with the original images, with a dimensionality of 256x256. The improvement in accuracy was present but not as tangible as expected. More interesting information can be found in the following section.

3.3. LeNet

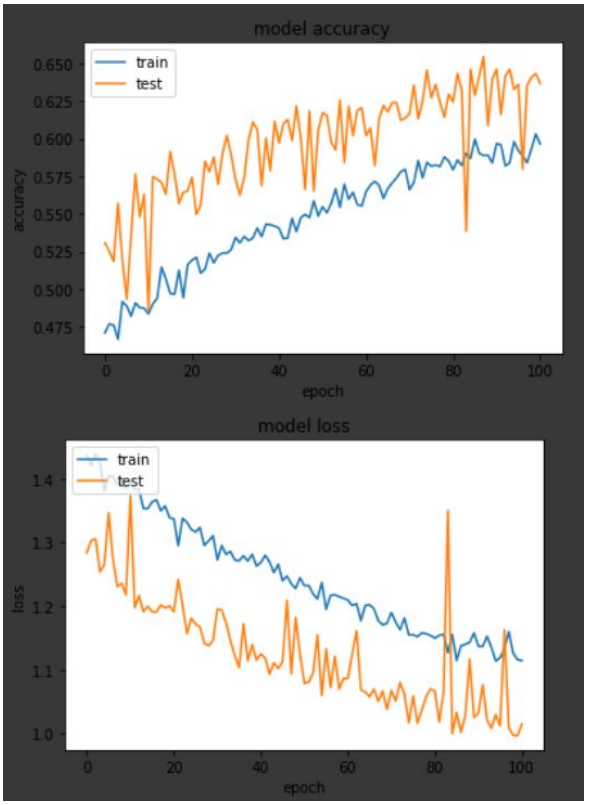
The training of this model has seen versions of it before reaching a good model. Considering the nature and simplicity of the model, the first attempt that I gave resulted in very bad results, not surpassing 0.2 of accuracy. I then decided to change the activation functions of the model, going from "tanh" to "relu" in order to make the decision of the model a bit more drastic. I have also proceeded to change the optimizer to SGD since it was suggested for simpler models such as this one.

This time as we can see in the graph results were much more promising, mostly since we were able to provide a somewhat linear improvement over the epochs. We can also notice that there was no sign of overfitting since the val_accuracy was always higher than normal accuracy. Analyzing the results on the other hand I have seen that certain classes were not in a good enough state, therefore I decided to push for another 100 epoch to see what kind of improvement I would have gained. This time training was much more unstable but still no sign of overfitting and as expected our test results improved a lot considering the distance from the training set. Classification also improved as we will be able to see in the next chapter. In order to conclude the tests with this Model I have decided to try 2 other training sessions. I have tried pushing to 300 epochs the model, showing how the model had reached almost its limits, also showing towards the end signs of overfitting. Finally I wanted to also test with a simpler model like this the result of adding data augmentation. Turns out that without it, we diverge to total overfitting around 50 epochs, making this model unusable with only the original dataset.

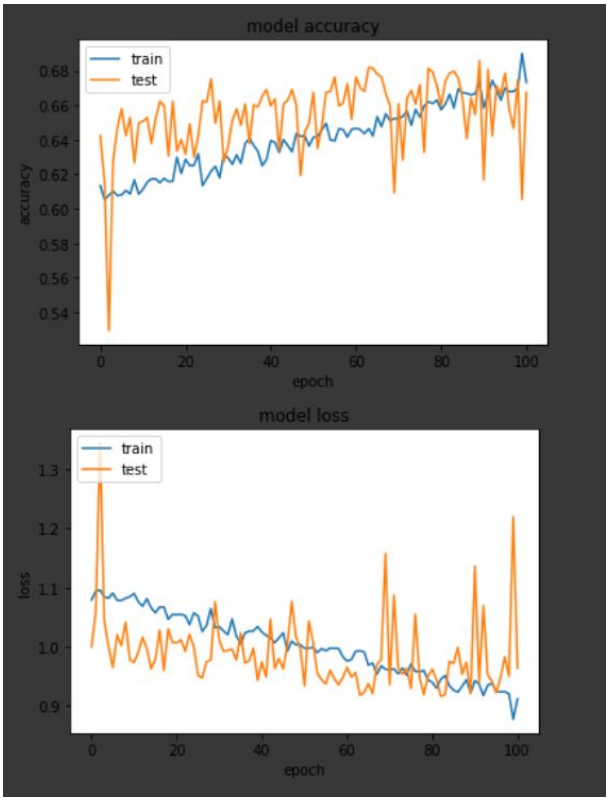
LENET - 128X128 - ADAM - 100 EPOCH



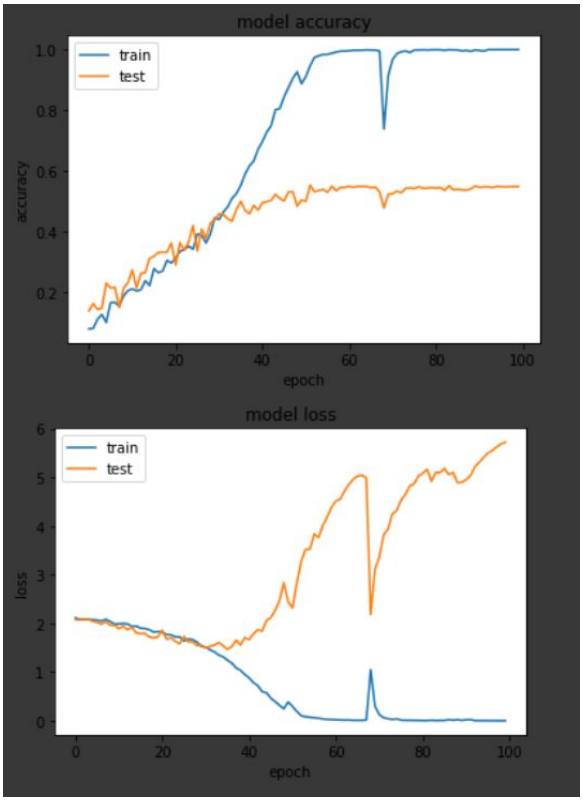
LENET - 128X128 - ADAM - 200 EPOCH



LENET - 128X128 - ADAM - 300 EPOCH



LENET - 128X128 - NO AUG - 100 EPOCH



4. Classification & Classification Result Analysis

4.1. LeNet

As mentioned in the training section, the first 100 epochs return an overall accuracy that is acceptable considering the Model, but classes such as Honey or Grocery Bag are way too low to be acceptable.

On the other hand the second training up to 200 epochs almost doubles the results, while also improving all the other classes.

Finally we can see how the final stretch to 300 epoch brings a downgrade, not making it worth it.

We can also see the same results also in our confusion Matrices below.

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 9s 307ms/step
                          precision    recall  f1-score   support

              Blueberries      0.603     0.650     0.626       220
             Energy_Drink      0.564     0.378     0.453       246
                    Honey      0.301     0.216     0.252       259
             Latex_Gloves      0.574     0.422     0.486       211
Vegetable_Chips_&_Crisps       0.352     0.821     0.493       240
             accent_plate      0.854     0.790     0.820       214
              grocery_bag      0.400     0.146     0.214       219
            plastic_knife      0.664     0.755     0.706       212

                 accuracy                          0.516      1821
                macro avg      0.539     0.522     0.506      1821
             weighted avg      0.531     0.516     0.498      1821
```

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 9s 305ms/step
                          precision    recall  f1-score   support

              Blueberries      0.888     0.614     0.726       220
             Energy_Drink      0.573     0.801     0.668       246
                    Honey      0.638     0.313     0.420       259
             Latex_Gloves      0.583     0.749     0.656       211
Vegetable_Chips_&_Crisps       0.540     0.738     0.623       240
             accent_plate      0.910     0.846     0.877       214
              grocery_bag      0.612     0.525     0.565       219
            plastic_knife      0.797     0.797     0.797       212

                 accuracy                          0.666      1821
                macro avg      0.692     0.673     0.666      1821
             weighted avg      0.687     0.666     0.659      1821
```

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 9s 295ms/step
                          precision    recall  f1-score   support

              Blueberries      0.665     0.759     0.709       220
             Energy_Drink      0.511     0.760     0.611       246
                    Honey      0.576     0.440     0.499       259
             Latex_Gloves      0.657     0.626     0.641       211
Vegetable_Chips_&_Crisps       0.668     0.546     0.601       240
             accent_plate      0.803     0.855     0.828       214
              grocery_bag      0.594     0.475     0.528       219
            plastic_knife      0.786     0.764     0.775       212

                 accuracy                          0.648      1821
                macro avg      0.658     0.653     0.649      1821
             weighted avg      0.653     0.648     0.644      1821
```

```
[[143  12   5   1  47   4   5   3]
 [ 35  93  33  16  47   1   2  19]
 [  5  13  56  14 141   5  11  14]
 [ 16  10  19  89  36   4  16  21]
 [ 10   4  22   1 197   1   3   2]
 [  8   4   2   3   6 169  10  12]
 [ 18  16  40  17  78   8  32  10]
 [  2  13   9  14   7   6   1 160]]
```

100 epochs

```
[[167  24   2   6  11   3   4   3]
 [ 15 187   9   9   7   2   7  10]
 [ 16  39 114   6  39  12  26   7]
 [  6  25  10 132   1   5  15  17]
 [ 22  34  40   2 131   2   8   1]
 [  6   6   1   6   1 183   7   4]
 [ 17  32  19  23   6  16 104   2]
 [  2  19   3  17   0   5   4 162]]
```

200 epochs

```
[[135  39   2   8  23   2   8   3]
 [  5 197   5  12  15   0   5   7]
 [  2  33  81  19  92   4  22   6]
 [  0  19   3 158   5   1  12  13]
 [  4  25  16   9 177   1   7   1]
 [  1   4   3   6   1 181  13   5]
 [  5  15  14  41  14   7 115   8]
 [  0  12   3  18   1   3   6 169]]
```

300 epochs

4.2. VGG16

How we have anticipated in the training section applying transfer learning brings amazing results in very few epochs. In fact the first image shows the results of just 25 epochs of training *(reached through early stopping)*.

On the other hand the second image shows that we gain no tangible benefit from reaching 100 epoch, since no class benefits from such extra work.

The third image shows the result of having SGD as our optimizer, which indeed leads to a worse classification as we theorized in the training section.

Finally we have our final results gathered from providing the network with 256x256 images, making classification for all classes very good, with the worse ones probably caused by the noise that we have asserted is present in our dataset.

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 10s 327ms/step
                          precision    recall  f1-score   support

              Blueberries       0.838     0.868     0.853       220
             Energy_Drink       0.796     0.825     0.810       246
                    Honey       0.688     0.537     0.603       259
             Latex_Gloves       0.734     0.863     0.793       211
Vegetable_Chips_&_Crisps        0.705     0.858     0.774       240
             accent_plate       0.954     0.864     0.907       214
              grocery_bag       0.790     0.603     0.684       219
            plastic_knife       0.809     0.896     0.850       212

                 accuracy                          0.784      1821
                macro avg       0.789     0.789     0.784      1821
             weighted avg       0.786     0.784     0.780      1821
```

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 10s 342ms/step
                          precision    recall  f1-score   support

              Blueberries       0.930     0.786     0.852       220
             Energy_Drink       0.880     0.744     0.806       246
                    Honey       0.532     0.776     0.631       259
             Latex_Gloves       0.736     0.858     0.792       211
Vegetable_Chips_&_Crisps        0.740     0.808     0.773       240
             accent_plate       0.964     0.879     0.919       214
              grocery_bag       0.863     0.635     0.732       219
            plastic_knife       0.935     0.816     0.872       212

                 accuracy                          0.786      1821
                macro avg       0.823     0.788     0.797      1821
             weighted avg       0.816     0.786     0.793      1821
```

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 10s 327ms/step
                          precision    recall  f1-score   support

              Blueberries       0.744     0.845     0.791       220
             Energy_Drink       0.566     0.907     0.697       246
                    Honey       0.614     0.375     0.465       259
             Latex_Gloves       0.556     0.782     0.650       211
Vegetable_Chips_&_Crisps        0.668     0.521     0.585       240
             accent_plate       0.913     0.879     0.895       214
              grocery_bag       0.707     0.484     0.575       219
            plastic_knife       0.911     0.769     0.834       212

                 accuracy                          0.688      1821
                macro avg       0.710     0.695     0.687      1821
             weighted avg       0.704     0.688     0.680      1821
```

```
Loaded 1821 test samples from 8 classes.
29/29 [==============================] - 28s 953ms/step
                          precision    recall  f1-score   support

              Blueberries       0.809     0.923     0.862       220
             Energy_Drink       0.809     0.894     0.849       246
                    Honey       0.751     0.710     0.730       259
             Latex_Gloves       0.923     0.791     0.852       211
Vegetable_Chips_&_Crisps        0.852     0.812     0.832       240
             accent_plate       0.969     0.888     0.927       214
              grocery_bag       0.764     0.813     0.788       219
            plastic_knife       0.911     0.920     0.915       212

                 accuracy                          0.841      1821
                macro avg       0.848     0.844     0.844      1821
             weighted avg       0.845     0.841     0.841      1821
```

5. Conclusion

To conclude and summarize the results that we have obtained, we can say that the best strategy for this kind of Dataset is definitely the one of Transfer Learning. It would be possible to consider several other networks and maybe consider which one returns the best results in order to later perform some fine tuning if necessary.
On the other hand, due to the nature of the Dataset, even a smaller Network such as LeNet, under the condition of providing sufficient Data augmentation is capable of reaching acceptable results. The advantage that the LeNet Network might have compared to VGG is that it's definitely lighter and also it's weights are considerably smaller, 26mb compared to 150mb (even though the difference is also due to the different dimensionality). Finally AlexNet is not necessarily a bad candidate for this task, but considering the Dataset and it's default parameters it wasn't able to reach a good enough result.

Finally possible improvements to this work could be found surely by cleaning the dataset from all its noise and trying to rebuild a transfer learning model to see how much it improves its results.

Rome, 23.12.2020
George Ciprian Telinoiu