SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning a.y. 22-23

Homework 2: Report

Hazem Dewidar - dewidar.1883881@studenti.uniroma1.it

January 8, 2023

1 Introduction

This homework requires us to choose a classification problem, building two CNN architectures and evaluate them. The chosen classification problem is a fruit classification problem. The dataset contains 10 classes of fruits and was is balanced. It's is available on Kaggle and can be downloaded from the link in the notebook. The dataset was already split into train and test sets. The train set was further split into train and validation sets.

1.1 CNN

A CNN (Convolutional Neural Network) is a type of neural network specifically designed to process data that has a grid-like topology, such as an image. CNNs are particularly useful for image classification and recognition tasks, as they are able to automatically learn features from the input data that are useful for classifying the images. They do this by applying filters to the input data, which allow the network to learn about different aspects of the image (such as edges, shapes, patterns) and use this information to make predictions. There are typically two stages in a CNN: the convolutional stage and the fully connected (or dense) stage.

- The convolutional stage involves applying a set of filters to the input data to extract features. These filters slide over the input data, performing element-wise multiplication and summing the results to produce a new feature map. This process is repeated for multiple filters to extract a set of different features from the input data. The filters are typically smaller in size than the input data, and are moved over the input data in increments known as strides. The size of the filters and the strides used can be varied to control the size of the output feature maps.

- The fully connected stage involves taking the output of the convolutional stage and flattening it into a single vector, which is then passed through a series of fully connected (dense) layers. These layers consist of nodes that are connected to all nodes in the previous layer, and they apply a linear transformation to the input data followed by a non-linear activation function. The output of the fully connected stage is a set of predictions, which are used to classify the input data.

The convolutional stage is typically responsible for extracting features from the input data, while the fully connected stage is responsible for using these features to make predictions.

1.2 The dataset

The dataset has 10 classes of fruits (see fig 7) , which are:

- Apple
- Banana
- Avocado
- Cherry
- Kiwi
- Mango
- Orange
- Pineapple
- Strawberries
- Watermelon

The dataset was balanced between all classes. It was already split into test and training sets. The training set was split again in post processing into a training set and validation set.

Here a brief description of what those sets are used for:

- A **training set** is a dataset used to fit the parameters (weights and biases in neural networks) of a model.
- A **validation set** is a dataset used to tune the parameters (architecture, not weights) of a model, for example to choose the number of hidden units in a neural network.
- A **test set** is a dataset used to evaluate the performance of a model on unseen data.

In general, the training set is used to build the model, the validation set is used to fine-tune the model and the test set is used to evaluate the performance of the model. The test set is generally held out until the very end of the model building process, and is used as a final check to ensure that the model is generalizable to unseen data.

To see the distribution among the sets, see fig 1.

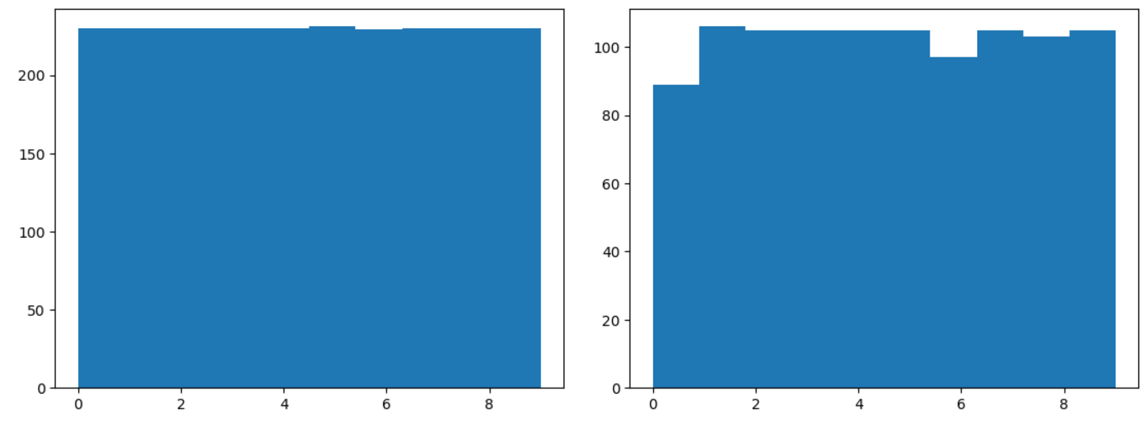

Figure 1: Sample distribution. A) Training set B) Test Set

2 Models and data preprocessing

In this section I will discuss the different models that I've used and give a brief explanation to how they work.

2.1 The architectures

2.1.1 CNN layers: what are they and how do they work

To create a CNN, we need to various layers, each one having a different purpose

Conv2D It applies a set of filters to the input data and produces a set of output maps. Each filter is a small matrix that slides over the input data, performing a dot product between the entries of the filter and the input data. The result of the dot product is summed up and added to a bias term to produce a single output value. The process is repeated for each position of the filter, producing an output map.

The formula for a single output value in a Conv2D layer is:

$$(f * g)(x, y) = \sum_m \sum_n K(m, n)I(x + m, y + n)$$

MaxPolling It takes the maximum value in a region. It does this by dividing the input data into a grid of smaller regions and taking the maximum value from each region. This helps to reduce the computational complexity of the model and also helps to reduce overfitting.

Dropout Dropout is a regularization technique that is used to prevent overfitting in a neural network. It works by randomly setting a fraction of the input units to zero during training. This forces the model to rely on a diverse set of features, rather than just a few strong ones, which helps to improve generalization.

Figure 2: Some examples from the dataset

Fully Connected A fully connected layer is a type of layer in which all of the nodes in the previous layer are connected to all of the nodes in the current layer. This means that each node in the current layer receives input from every node in the previous layer, and produces an output that will be passed on to the next layer (if there is one)

2.1.2 Callback

Early stopping is a technique used to stop the training of a machine learning model before the number of epochs is reached. It is based on the idea of monitoring the model's performance on a validation set for each epoch and stopping the training when the performance stops improving or starts deteriorating.

ReduceLROnPlateau educes the learning rate of the optimizer when the model's performance stops improving on the validation set

2.2 Classifiers

Model 1 The first CNN is a simple one. It has the following architecture:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896


 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )
 dropout (Dropout)           (None, 111, 111, 32)      0

 batch_normalization (BatchN  (None, 111, 111, 32)     128
 ormalization)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 54, 54, 64)        0

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 26, 26, 128)       0

 conv2d_3 (Conv2D)           (None, 24, 24, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 128)      0
 2D)

 dropout_3 (Dropout)         (None, 12, 12, 128)       0

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 512)               9437696

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 9,683,786
Trainable params: 9,683,722
Non-trainable params: 64
_________________________________________________________________
```

There are four convolution layers, each having a conv2d, maxPolling (to reduce the size of the image) and a dropout layer. The dropout layer is used to prevent overfitting. Only the first layer has batch normalization layer added to it. After the convolution phase, the image is flattened and then passed through 2 dense layers. The first dense layer has 512 neurons and the second dense layer has 10 neurons. The last layer has softmax activation function to give the probability of each class and has 10 neurons as there are 10 classes.

Model 2 The second model uses transfer learning. It uses the VGG16 model and adds a dense layer with 512 neurons and a dense layer with 10 neurons. The last layer has softmax activation function to give the probability of each class and has 10 neurons as there are 10 classes.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 21,140,042
Trainable params: 6,425,354
Non-trainable params: 14,714,688
_________________________________________________________________
```

VGG16 is a convolutional neural network model that was used to win the ILSVRC achieving state-of-the-art performance on the image classification task. It is composed of 16 convolutional layers and 3 fully connected layers, and has a total of 138 million parameters. The model is trained on the ImageNet dataset (it has 1000 classes and around 1M images). VGG16 has been widely used as a base model for various vision tasks and has been successfully applied to many tasks, including object recognition, face detection. It has also been used as a feature extractor for tasks such as object detection and fine-grained image classification. In the original dataset, not all fruits that are in the dataset used in this homework are present.

Model 3 The third model has three convectional layers and three dense layers. It has the following architecture:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 128)      0
 2D)

 dropout (Dropout)           (None, 26, 26, 128)       0

 conv2d_3 (Conv2D)           (None, 24, 24, 128)       147584

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 12, 12, 128)       0

 flatten (Flatten)           (None, 18432)             0

 dense (Dense)               (None, 64)                1179712

 dense_1 (Dense)             (None, 16)                1040

 dense_2 (Dense)             (None, 10)                170

=================================================================
```

The point of this model was testing what happens if we use two small dense layers instead of a big dense layer.

Model 4 The last model is a model in which I've introduced a regulation with a weight decay regularizator. With this model, I've played with two hyper-parameters: the learning rate and the weight decay (see section 3.4)

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_8 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_9 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_9 (MaxPooling  (None, 54, 54, 64)       0
 2D)

 conv2d_10 (Conv2D)          (None, 52, 52, 128)       73856

 max_pooling2d_10 (MaxPoolin  (None, 26, 26, 128)      0
 g2D)

 dropout_4 (Dropout)         (None, 26, 26, 128)       0

 conv2d_11 (Conv2D)          (None, 24, 24, 128)       147584

 max_pooling2d_11 (MaxPoolin  (None, 12, 12, 128)      0
 g2D)

 dropout_5 (Dropout)         (None, 12, 12, 128)       0

 flatten_2 (Flatten)         (None, 18432)             0

 dense_6 (Dense)             (None, 64)                1179712

 dense_7 (Dense)             (None, 16)                1040

 dense_8 (Dense)             (None, 10)                170

=================================================================
```

3 Experiments

In this section I will show the plots and the results obtained. On one a model, I've changes some hyper-parameters (for more details, see the subsection 3.4). To train the CNN, I had to use categorical labels. Here is the association Fruit : index

'apple': 0, 'banana': 1, 'avocado': 2, 'cherry': 3, 'kiwi': 4, 'mango': 5, 'orange': 6, 'pinenapple': 7, 'strawberries': 8, 'watermelon': 9

3.1 Model 1

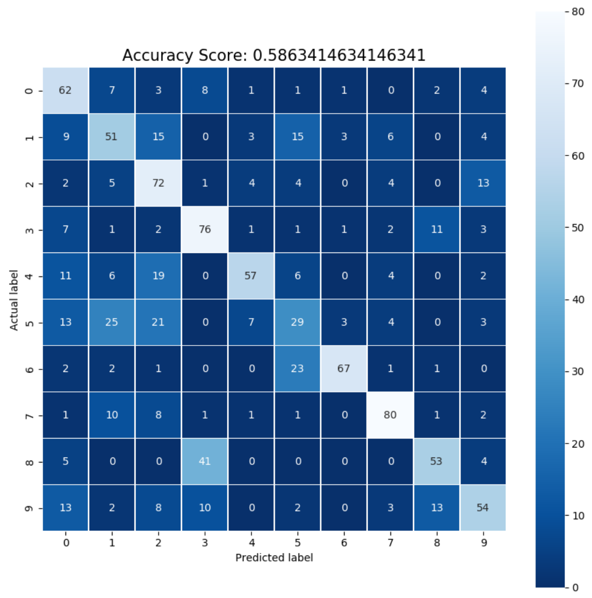

Figure 3: Confusion matrix for the model 1

```
              precision    recall  f1-score   support

       Apple       0.50      0.70      0.58        89
      Banana       0.47      0.48      0.47       106
     avocado       0.48      0.69      0.57       105
     cherry        0.55      0.72      0.63       105
        kiwi       0.77      0.54      0.64       105
       mango       0.35      0.28      0.31       105
      orange       0.89      0.69      0.78        97
  pinenapple       0.77      0.76      0.77       105
strawberries       0.65      0.51      0.58       103
  watermelon       0.61      0.51      0.56       105
```

```
    accuracy                          0.59      1025
   macro avg       0.60      0.59      0.59      1025
weighted avg       0.60      0.59      0.59      1025
```

Overall, the model has an accuracy of 0.59, which means that it made correct predictions in 59% of the cases. The macro average of the precision, recall, and f1-score was 0.60, while the weighted average was 0.59.

The model performed well in some classes (orange and pineapples with a precision of 0.89 and 0.77 and a high f1-score). However, it performed poorly in other classes, such as mango and strawberries, with a precision of 0.35 and 0.65 respectively, and a low f1-score.

The model's performance could be improved by increasing the precision and recall for the classes in which it performed poorly. Maybe increasing the overall accuracy of the model would likely improve its performance.

3.2 Model 2

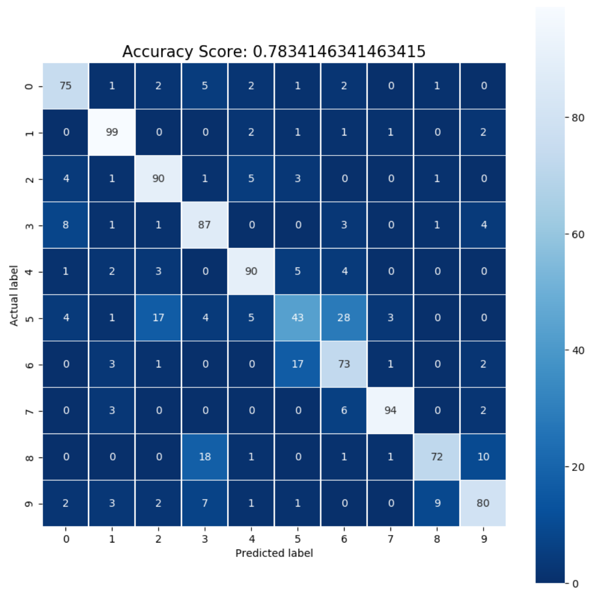

Figure 4: Confusion matrix for the model 2

```
              precision    recall  f1-score   support


       Apple       0.80      0.84      0.82        89
      Banana       0.87      0.93      0.90       106
```

```
     avocado       0.78      0.86      0.81       105
      cherry       0.71      0.83      0.77       105
        kiwi       0.85      0.86      0.85       105
       mango       0.61      0.41      0.49       105
      orange       0.62      0.75      0.68        97
  pinenapple       0.94      0.90      0.92       105
strawberries       0.86      0.70      0.77       103
  watermelon       0.80      0.76      0.78       105

    accuracy                           0.78      1025
   macro avg       0.78      0.78      0.78      1025
weighted avg       0.78      0.78      0.78      1025
```

This is the best model used in this homework. It's not surprising considering the fact that the VGG16 has a deeper architecture than what I could train (due to computational power).

The model performed with an accuracy of 0.78 The model performed well in most classes, with a high precision recall and f1-score. Classes such as banana and pineapples had a precision of 0.87 and 0.94 respectively, and a high f1-score. However, some classes, such as mango and orange, had a lower precision and recall, with a precision of 0.61 and 0.62 respectively, and a lower f1-score. In general, the model's performance was good, with a high accuracy and a high average of the precision, recall, and f1-score. However, there is still room for improvement, particularly in the classes that had a lower precision and recall.

3.3 Model 3

```
              precision    recall  f1-score   support

       Apple       0.46      0.64      0.54        89
      Banana       0.40      0.37      0.38       106
     avocado       0.53      0.49      0.50       105
      cherry       0.53      0.63      0.58       105
        kiwi       0.48      0.61      0.54       105
       mango       0.44      0.35      0.39       105
      orange       0.71      0.79      0.75        97
  pinenapple       0.65      0.75      0.70       105
strawberries       0.59      0.49      0.53       103
  watermelon       0.59      0.28      0.38       105

    accuracy                           0.54      1025
   macro avg       0.54      0.54      0.53      1025
weighted avg       0.54      0.54      0.53      1025
```

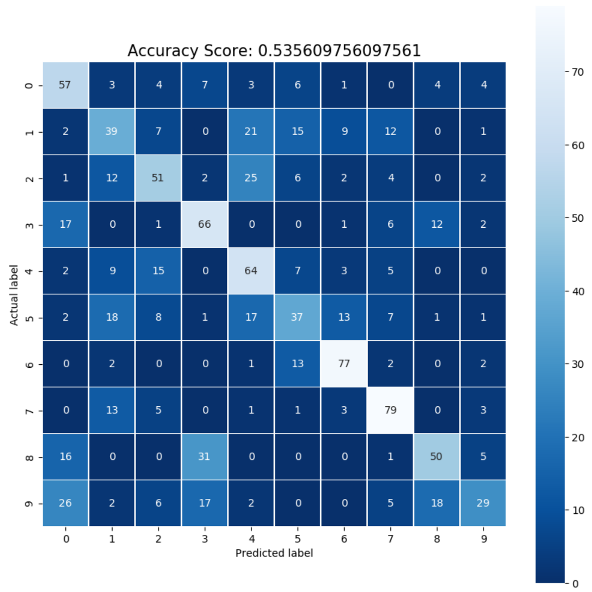

Figure 5: Confusion matrix for the model 3

The model performed with an accuracy of 0.54. The model performed poorly in most classes, with a low precision and recall, and a low f1-score. Classes such as banana, mango, watermelon, and strawberries had a precision and a low f1-score.

3.4 Model 4

The difference in these models are in hyper parameters. In the first model I used:

- Learning rate: 0.0001
- Weight Decay: 1e-4

In the second model:

- Learning rate: 0.00001
- Weight Decay: 1e-2

3.4.1 Model 4.1

```
              precision    recall  f1-score   support

       Apple       0.47      0.58      0.52        89
      Banana       0.52      0.42      0.46       106
     avocado       0.46      0.50      0.48       105
```

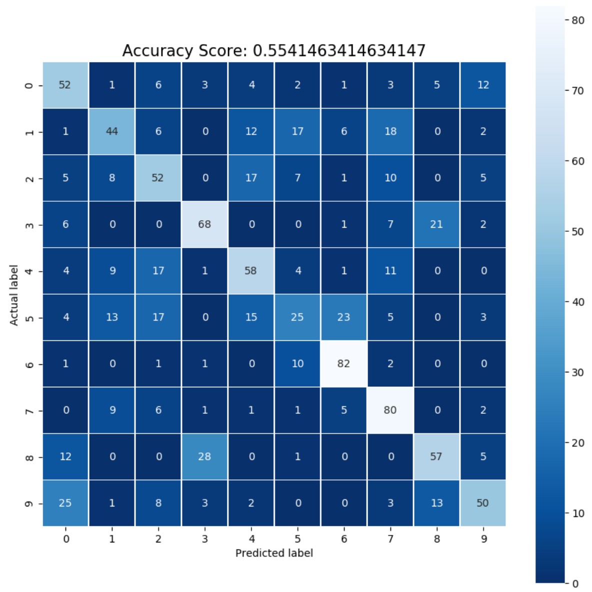

Figure 6: Confusion matrix for the model 4

```
      cherry       0.65      0.65      0.65       105
        kiwi       0.53      0.55      0.54       105
       mango       0.37      0.24      0.29       105
      orange       0.68      0.85      0.76        97
  pinenapple       0.58      0.76      0.66       105
strawberries       0.59      0.55      0.57       103
  watermelon       0.62      0.48      0.54       105

    accuracy                           0.55      1025
   macro avg       0.55      0.56      0.55      1025
weighted avg       0.55      0.55      0.54      1025
```

3.4.2 Model 4.2

```
              precision    recall  f1-score   support


       Apple       0.43      0.25      0.31        89
      Banana       0.24      0.07      0.10       106
     avocado       0.27      0.80      0.41       105
      cherry       0.45      0.67      0.54       105
```

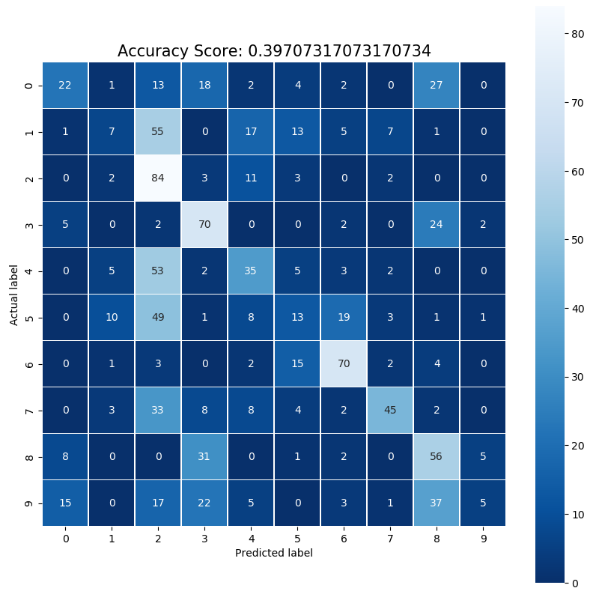

Figure 7: Confusion matrix for the model 4.2

```
        kiwi       0.40      0.33      0.36       105
       mango       0.22      0.12      0.16       105
      orange       0.65      0.72      0.68        97
  pinenapple       0.73      0.43      0.54       105
strawberries       0.37      0.54      0.44       103
  watermelon       0.38      0.05      0.08       105

    accuracy                           0.40      1025
   macro avg       0.41      0.40      0.36      1025
weighted avg       0.41      0.40      0.36      1025
```

Comparing the two classifiers, it is clear that the first classifier is performing better than the second classifier. We can obeserve that higher learning rate and lower weight decay can lead to have better performance but probabily it led to overfitting. On the other hand, a lower learning rate and higher weight decay can prevent overfitting, but can also lead to slower convergence and poorer performance.

4 Conclusions

To summarize, this was not an easy task. Although it may seem trivial, fruits recognition is not a trivial task because there are many different types of fruits, they can vary in terms of size, shape, color, texture, and so on. In the dataset, there were fruits sliced so the shape or the color weren't a strong features (an avocado is brown but inside it's green) For example, apples can be red, green, or yellow and can vary in size and shape, bananas can be yellow, green, or brown and can be straight or curved, and avocados can be round or pear-shaped and can have smooth or rough skin. In addition, some fruits may have similar visual characteristics, making it challenging to distinguish between them. For example, cherries and strawberries are both small and red, and kiwis and pineapples both have rough, textured skin. As a result, it is not easy for a computer to accurately recognize and classify different types of fruits. It may be interesting to see if increasing the number of convolutions layers (to extract more features) will improve performance. To train the model, I've used both Google Colab and my own PC but often I would get a resource exhaust error in both platforms. The best model has obtained around 78% of accuracy