Simone Quadrelli

Fruit classification

Academic year 2018 - 2019

Contents

List of Tables

List of Figures

Abstract

The aim of the project is to investigate the ability of K-nearest neighbours (KNN), support vector machines (SVM) and convolutional neural networks (CNN) to classify images of fruits and vegetables using as feature vectors the images rgb and grayscale representation.
The analysis develops following some major steps: first features are extracted, then feature reduction is performed. Hence, KNN and SVM performance are computed with different parameters and number of features. In the end, CNN are applied to the whole features.
The most remarkable findings concern CNNs, which perform the best, however KNN reaches a surprisingly high accuracy. It is also worth noticing that even if grayscale features are very useful, rgb features are more meaningful.

1 Introduction

The objective of this project is to classify a variety of fruits and vegetables whose images are stored in a recently released dataset.
Fruit and vegetable classification is a challenging task; indeed, there are two different kind of problems that must be overcome: vegetables and fruits within the same class can differ in shape and colour due to maturity and growth. On the countrary, vegetables and fruits in different classes, such as red apples or grapes, may be very similar both in textures and in colours. Fruit and vegetables classification has a key role in many industries: it can be exploited to make agriculture more autonomous than ever before in history, but it can also be applied to all stages of the supply chain to assess the quality of the products from the producer to the consumer. The ability of harvesting robots can be enhanced by enforcing fruit and vegetables recognition, indeed it may allow them to recognize not only the type of fruit or vegetable but also its maturity or whether it is rotten. Moreover, autonomous classification provides more consistent quality assessment since it does not depend of the opinion of different workers. [2]
The projects analyzes the classification performance of three machine learning algorithms on different fruits. The k-nearest neighbour (KNN) algorithm was used to compute a baseline, a more sophisticated one, the support vector machine (SVM) was later used for classification. The last approach was to construct a convolutional neural network (CNN) because they are known to suit well image classification problems.
The are two different kind of features that are extracted to feed the algorithms: the grayscale and the rgb representation of the images.

The objective of the project are:

1. Analyze the accuracy result achieved with grayscale features.
2. Study the improvement in performance induced by rgb features.
3. Compare the performance of KNN with that of more sophisticated algorithms.

2 Dataset and Features

The analyzed dataset, known as *Fruits 360*, was originally published in 2017 by Horea Muresan and Mihai Oltean to address fruit and vegetable classification [1]. The existence of either too

small datasets or too low-quality dataset was the most compelling problem. On the countrary, this dataset provides more that 70000 images of 114 different fruits and vegetables. Each image has a definition of 100 $\times$ 100 pixels. To further increase the reliability of the results, the authors decided to enrich it with images of rotated fruits and vegetables. However, those images make the training process much harder.
As the authors explain, the dataset was produced as follows: fruits and vegetables were planted in the shaft of a low speed motor (3 rpm) and a short movie of 20 seconds was recorded. However, due to the variations in the lighting conditions, the background was not uniform and a dedicated algorithm was exploited to extract the fruit from the background. Hence, the images were of such high quality that no pre-processing was required. It is also worth noticing that the dataset was already split in train and test set, therefore no further splitting was required.
Fruits and vegetables have several distinct visual characteristics that can be extracted from their pictures. Colour, shape, size and texture are the most commonly used features[1] in image classification. Among all the possible features that can be extracted from an image, colour features play a key role in fruit and vegetable classification. Indeed, they seem to be suitable for the problem and their computation is very fast and efficient [2]. Thus, the project focuses on the grayscale and rgb representation of the images.
The following lines introduce the reader to some concepts of colour spaces and colour representation. Colour spaces are a specific organization of fundamental colours the allows to reproduce a wide variety of colours and shades of gray. Two different colour spaces are exploited in this project: grayscale and rgb.

2.1 Rgb colour space

The rgb colour model is an additive colour model, meaning that red, green and blue are added together to reproduce a wide array of colors. A color in the rgb colour model is described by indicating how much the red, green, and blue it includes. Therefore, the colour of a pixel is expressed as an rgb triplet (R,G,B) whose components can vary from 0 to 255, since each channel is represented by a byte.

2.2 Grayscale colour space

Since the dataset contains rgb images, it was necessary to extract a grayscale representation of the pixels to be used as features. Notice that in grayscale representation each pixel is represented by a byte, meaning that the possible values it can be in range from 0 to 255.
The conversion algorithm implemented by the library *imager* is the Luma algorithm. Each rgb channel R, G, B of a pixel x are passed to a function $\Gamma(x)$ to obtain its transformation, respectively R', G', B'.
The Γ function is defined as:

$$x' = \Gamma(x) = Ax^{\gamma} \tag{1}$$

where x in the pixel intensity, A is a scalar whose value is often 1 and γ is usually $\frac{1}{2.2}$.
To obtain the final grayscale representation y of a pixel x, the following linear transformation is applied:

$$y = 0.21R' + 0.71G' + 0.07B' \tag{2}$$

[3]

[1] Features are the characteristics of an object that can distinguish it from other objects.

3 Data analysis

This section provides the reader an insight of the procedure pursued to analyze the data, it also provides a commentary of the results.
The procedure described in the section is the same for both the grayscale features and the rgb features. Algorithm 1 provides a brief overview of the analysis.

Algorithm 1 Schema of the analysis

1: Feature extraction
2: Dimensionality reduction
3: Extraction of the most significant features
4: Construction of a baseline using KNNs
5: Evaluation of SVMs
6: Evaluation of CNNs

3.1 Feature extraction and dimensionality reduction

The first major step in the analysis is the data extraction and the computation of the data matrix and the vector of the labels concerning the following fruits: *Apple Golden 1*, *Apple Golden 2*, *Apple Red 2*, *Apple Red 1*, *Banana*, *Apricot*, *Blueberry*.
As previously stated, the dataset contains 100x100 pixels images, therefore the data matrix with rgb features has 409440000 elements. Bearing that in mind, the images are rescaled to 50×50 pixels to reduce the computational complexity of the problem. The feature vector of each image is extracted by the functions of the library *imager*. Once all the data vectors are extracted, a data matrix is created whose rows contain the linearized representation of the image. At the same time a vector of labels is computed.
Most learning algorithms are exponentially slow in the number of the feature or at least significantly slow down as the number of feature increases. Furthermore, it is reasonable to believe that many of the pixels are useless since they are just part of the background. As a result, Principal component analysis (PCA) is exploited to reduce the dimensionality of the data and select the most promising features. PCA was chosen because it is one of the most common technique to reduce the dimensionality of the data, since it is very efficient and many libraries implement it. It computes all the principal components that can be seen as the axis along which data differ the most.
It is also worth noticing that before applying PCA data points x are standardized in the following way:

$$\frac{x - \mu}{\sigma} \tag{3}$$

where μ is the mean value and σ is the standard deviation of the data.
When using the grayscale features the execution time required to compute the PCA is 5 minutes and 30 seconds. However, the execution time required to compute the PCA is about 57 minutes for rgb features[2].

[2]All time measurement rely on a Intel Core i5-7200U dual core processor and 8 GB DDR4 RAM

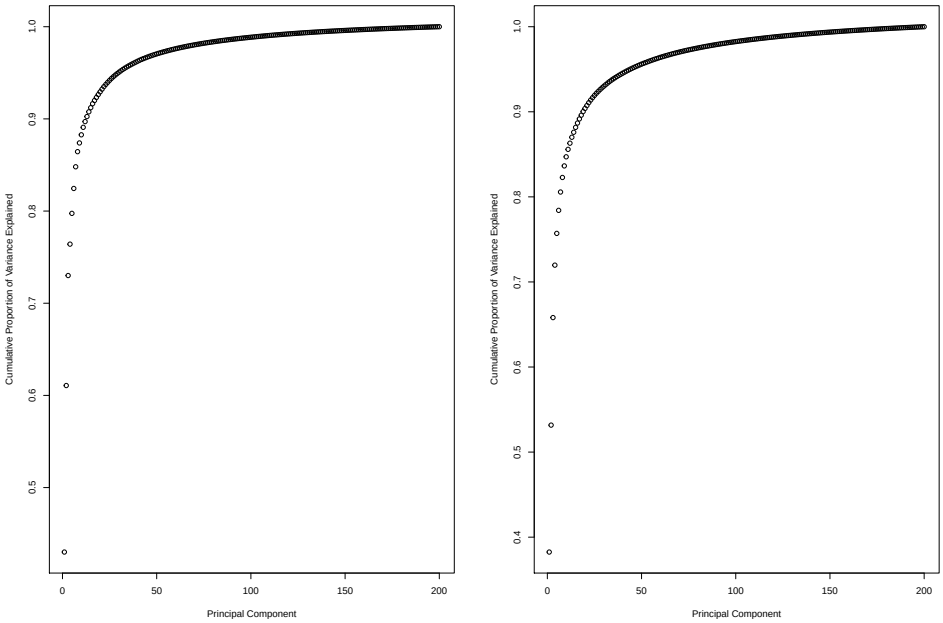

Figure 1: Cumulative variance explaind by the principal components

As figure 1 shows, in both cases, using 150 principal components is enough to explain all the variance in the data. However, having fixed the number of components, rgb components explain less variance than the grayscale components.
To compare and analyze the performance of the learning algorithms the following set of principal components $\{3, 10, 50, 100, 150\}$ are used. The choice is justified by the objective to select the number of components that should give the best result having at the same time a fast execution time.

3.2 Baseline and KNN

The k-nearest neighbours algorithm seems to be an appropriate choice to define a baseline because it is known to be one of the most efficient classification algorithm, even if it is very simple. Indeed, the KNN classifies a test point with the labels of the closest training point. Moreover, in image classification it seems to perform quite as well as more sophisticated algorithms such as support vector machines. [4]
To evaluate the performance of KNN and the other algorithms, the most natural choice is the accuracy, defined as:

$$\text{accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}. \tag{4}$$

The number of neighbours (k) chosen to construct the baseline are: $1, 3, 7, 9, 11$. As can be seen from figure 2, there is no remarkable difference in performance if the number of features exploited

is 10 or more. On the other hand, when only three features are used the algorithm achieves lower performances. This difference is even more sharp when rgb features are involved[3], indeed the accuracy lies about 0.10 below the other accuracy lines.
The best number of features to select is 50, since with all the possible value of k KNN performs best with 50 features.
If we limit our analysis, considering just the result of KNN with at least 10 features 1-NN has the very best results, even if up to 7 neighbours there is no significant difference in performance. Overall, KNN reaches a very significant accuracy: the accuracy level when considering rgb features reach up to 0.91, while the accuracy level reached using grayscale features is about 0.82. As expected the rgb features are more reliable and are associated with better results.
Overall, the best model is 1-NN with 50 features.

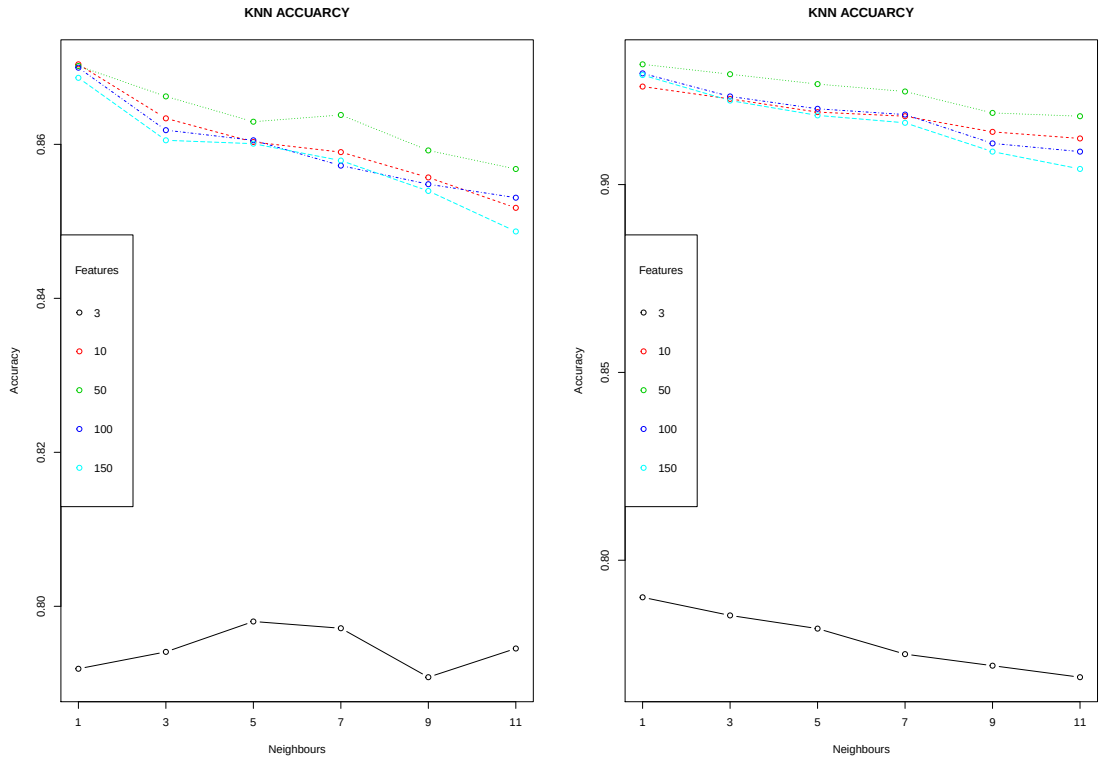

Figure 2: Test accuracy knn

The execution time of the algorithm is acceptable and does not exceed 30s. It is worth noticing that the execution time, given the number of features, have just negligible variation when the number of neighbours changes.

[3]The figure on the left always refers to the result obtained using the grayscale features. The figure on the right refers to the result obtained using the rgb features.

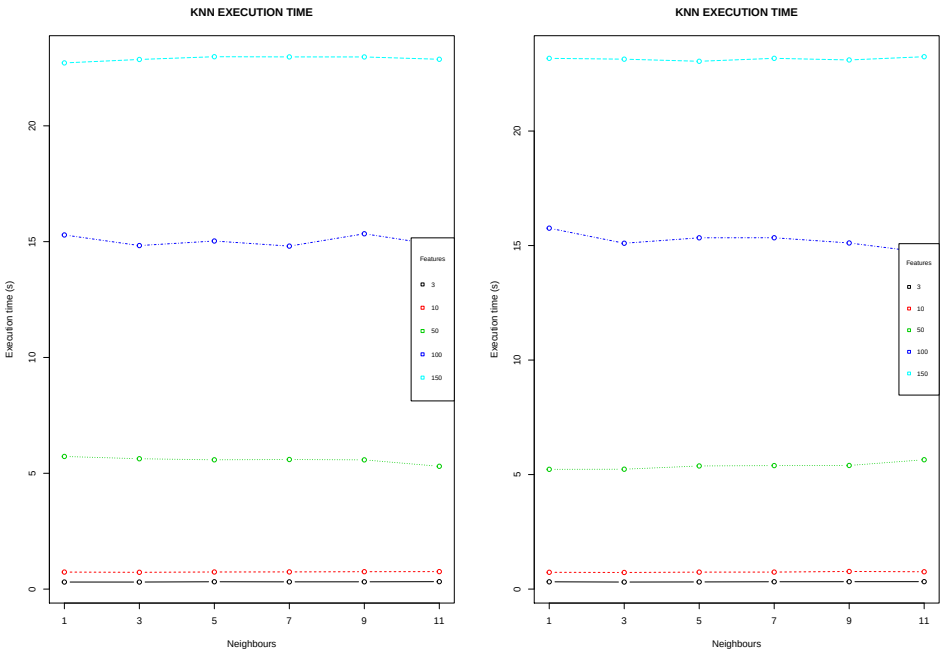

Figure 3: Execution time knn

Overall, the baseline is very promising but hard to improve. Indeed, as suggested by the literature, KNN seems to be very suitable to classify images.

3.3 Support Vector Machines

Support vector machines (SVM) are often used in image classification, even if in their simplest form they are just linear classifiers. However the dataset seems hardly linearly separable, thus the gaussian kernel was applied to allow generic curves to be used. The gaussian kernel is a linear combination of polynomial kernels that allows for more accurate predictions but slows down the algorithm.
The plot on the left of figure 4 shows the accuracy of the SVM on the grayscale features. The SVM reaches a lower accuracy with respect to the accuracy of KNN: it is less precise by 0.02/0.03. On the contrary, SVM with rgb features performs as well as the KNN algorithm.
In both cases the best accuracy, as before, is reached when the number of features is 50.

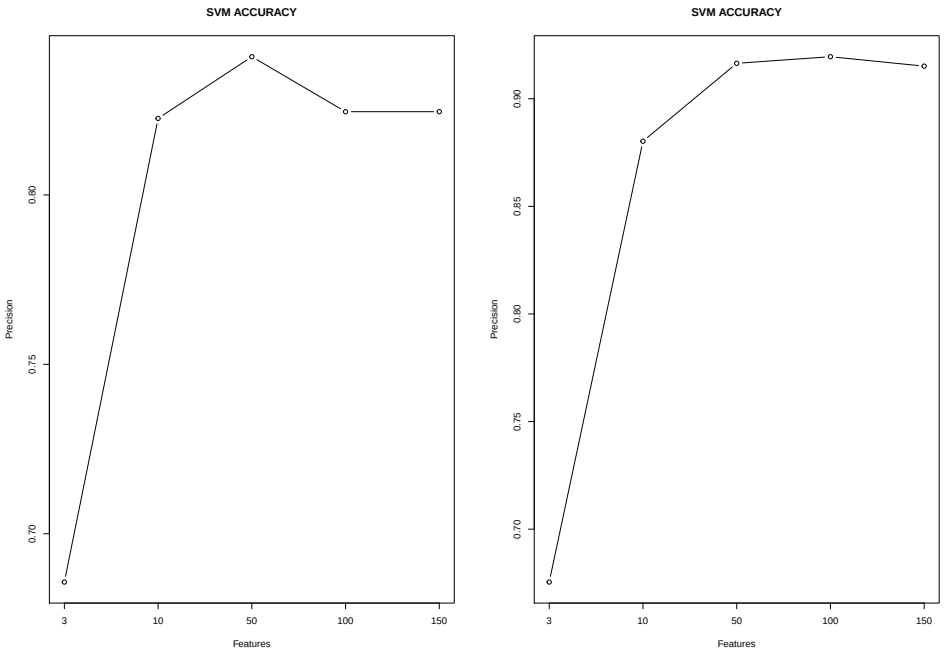

Figure 4: Test accuracy SVM

The execution time of SVMs is way higher than the execution time of the KNNs, however it remains reasonable as figure 5 shows.

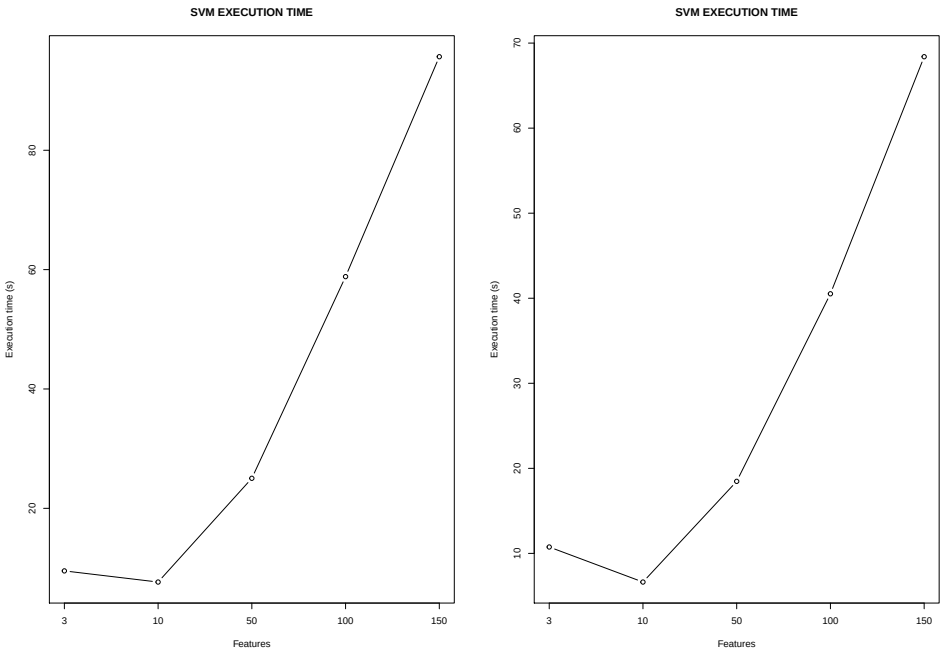

Figure 5: Execution time SVM

Overall, the SVM does not improve the result defined in the baseline that remains still very significant. Only a negligible improvement is reached when using the rgb features.

3.4 Convolutional neural network

Deep learning algorithms are a class of machine learning algorithms often used to solve classification problems by exploiting multiple layers to extract higher level features. They are built stacking some convolutional and pooling layers over which lay some dense layers[4]. Among the various deep learning algorithms, convolutional neural networks (CNN) are suitable for image classification since they can recognize translation invariant features.
The implementation of the CNNs relies on the *Keras* and *tensorflow* API, they both are very famous and widespread libraries provided by Google that underlay many famous software like DeepDream. Figure 6 shows the training accuracy of the CNN with respect to the number of epochs in the training process. The training accuracy reaches as high as 0.99 while the test accuracy is a bit lower for both grayscale and rgb features. When grayscale features are involved the test accuracy of the CNN is about 0.90, while is 0.94 when considering rgb features.

[4]Section 4.4 provides a wider explanation of CNNs

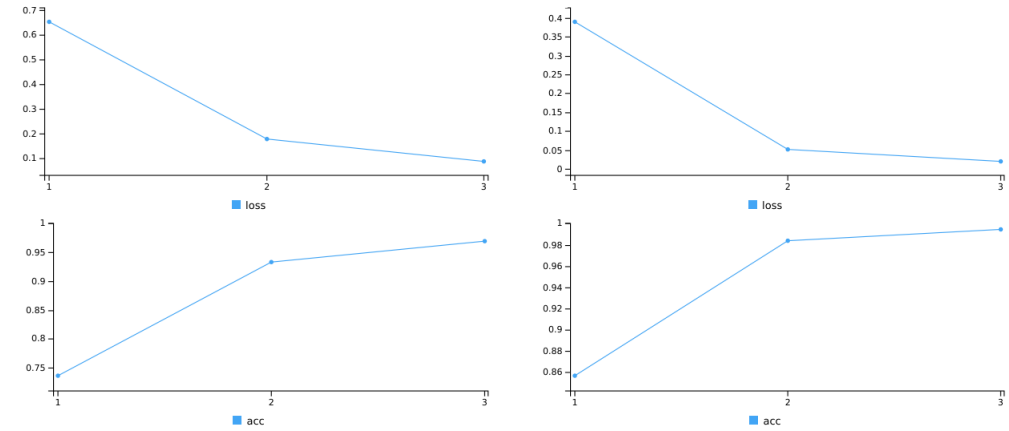

Figure 6: CNN training accuracy

The training time of both the nets is about 1 minute and the computation of the prediction requires about 30 seconds. Therefore CNN are faster than some instances of SVM but slower than the KNN. Overall, the CNNs seem to be the best model to classify images.

4 Mathematical background

This section provides the mathematical and algorithmic explanation of the algorithm used in the analysis.

4.1 PCA

This section introduces the mathematical background conserning the Principal component analysis (PCA).
The principal components of a set of data in R^n provide a sequence of best linear approximations to that data in R^d, where $d < n$. The principal components are computed solving an optimization problem than consists in finding two matrices: the compression matrix $W^{n\times d}$ and the recovery matrix $U^{d\times n}$. The former matrix maps the features vector from R^n in R^d, the latter recovers approximately the origianl data mapping from R^d in R^n. The matrices can be computed solving following optimization problem:

$$\underset{W\in R^{n\times d},\, U\in R^{n\times d}}{\operatorname{argmin}} \sum_{i=1}^{m} ||\mathbf{x}_i - UW\mathbf{x}_i||_2^2. \tag{5}$$

The optimization problem can be rewritten as:

$$\underset{U\in R^{d\times n}:\, U^TU=I}{\operatorname{argmin}} \sum_{i=1}^{m} ||\mathbf{x}_i - UU^T\mathbf{x}_i||_2^2 \tag{6}$$

because it was shown that U and W are orthonormal and $W = U^T$.
It is possible to apply the equality

$$||\mathbf{x}_i - UU^T\mathbf{x}_i||^2 = ||x||^2 - \text{trace}(U^T\mathbf{x}\mathbf{x}^TU) \tag{7}$$

to the objective function to come up with this new optimization problem

$$\underset{U \in R^{d\times n}: U^TU=I}{\text{argmax}} \text{trace}(U^T \sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^TU) \tag{8}$$

whose soluton is the matrix U. The columns of U are the eigenvectors of $A = \sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^T$ known as principal components. [6] [7]

4.2 KNN

KNN is algorithm that relies on a simple assumpltion: similar data vectors belong to the same class. Therefore, to measure similarity the notion of distance can be exploited. KNN works with all possible distances, however the euclidean distance ρ was used in the project:

$$\rho(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d}(\mathbf{x}_i - \mathbf{y}_i)^2} \tag{9}$$

where $\mathbf{x}$, $\mathbf{y} \in R^d$.
The algorithm just computes the distance between the test point and all the training points, then it selects the label of the first k closest points. The predicted class is the one with the highest frequence among the k closest points.

4.3 SVM

This section provides an overview of Support Vector Machines, in particular we focus on SVM for binary calssification since it allows math to be simpler and clearer. However, multiclass SVM are built using binary SVMs. The most common technique in practice has been to build a set one-versus-rest classifiers and to choose the class which classifies the test with greatest margin. Another strategy is to build a set of one-versus-one classifiers, and to choose the class that is selected by the most classifiers. [5]
Support Vector Machines is a model that determines the separating hyperplane $\mathbf{w}$ with the maximum margin between the closest positive labels and negative labels. The margin can be defined as:

$$\text{margin}(\mathbf{w}) = \min_{i=1,..,m} y_i\mathbf{w}^T\mathbf{x}_i \tag{10}$$

where y is the class, $\mathbf{x}$ is the data vector and m is the number of vectors used to train the model. The hyperplane found in such a way should reduce the probability of misclassification for points with different labels that are close to one another and close to the hyperplane.
On a linearly separable data, the SVM solves the following problem:

$$\begin{aligned} &\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{R}^d}{\text{argmin}} \frac{1}{2}\|\mathbf{w}\|^2 \\ &s.t.\ y_i\mathbf{w}^T\mathbf{x}_i \geq 1\ \forall i \in \{1, ..., m\} \end{aligned} \tag{11}$$

However data are hardly linearly separable, therefore SVM can be generalized to solve the problem on non-linearly separable data as follows:

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{R}^d}{\operatorname{argmin}} \left(\frac{1}{m} \sum_{i=1}^{m} h_i(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) \tag{12}$$

where $h(\mathbf{w})$ is the hinge loss. To further increase the classification capability of a SVM kernel functions can be applied. The gaussian kernel was applied in the implementation of SVM. The gaussian kernel can be defined as:

$$K_\gamma(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x})\psi(\mathbf{x}')^T = \exp\left(-\frac{1}{2\gamma} ||x - x'||^2 \right) \ \gamma > 0 \tag{13}$$

Therefore, the objective to minimize becomes

$$\mathbf{w}^* = \underset{\mathbf{w}\in\mathcal{R}^d}{\operatorname{argmin}} \left(\frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y\mathbf{w}\psi(\mathbf{x}_i)^T\} + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) \tag{14}$$

where y is the label of the data. [6] [7]

4.4 CNN

Deep learning algorithms are a class of machine learning algorithms often used to solve classification problems by exploiting multiple layers to extract higher level features. Among the various deep learning algorithms convolutional neural netwoks (CNN) are suitable for image classification since they can recognize translation invariant features. Moreover, they were insipired by the human eye and try to mimic it.
The topology of a CNN consists in a stack of convolutional and pooling layers followed by dense layers.
Convolutional layers optimize the matrices known as filters that are applied to each portion of the data. It is worth noticing that the name convolutional neural network origins form the filtering operation known as discrete convolution in mathematics.
The pooling layer extracts a reduced representation of the data computed in the convolutional layer. Indeed, in this layer the dimensionality of the data is reduced by selecting a subset of the image and extracting a single value from each subset; usually the highest, mean or minimum value in the subset is extracted.
The process can be repeated by stacking convolutional and pooling layers; then, some fully conneted layers are stacked upon them. The purpose of these layers is to compute the probability that the input image belongs to each class. Therefore, the last layer of the net has a number of neurons equal to the number of class and the output is computed exploiting the *softmax* function that provides a probability distribution. The prediction computed is the class having the highest probability.
The main reason to prefer deep learning techniques to generic neural networks is that many layers with fewer weights are more flexible and precise in classification than neural networks with many weights and fewer layers. CNNs are also very computationally efficient because they optimize only the small filters in the convolutional layers and just few dense layers whose optimization is computationally more expensive.

5 Conclusions

This work aims to analyze the performance of well-known classification algorithms: k-nearest neighbour, support vector machines and convolutional neural networks on *Fruit 360* dataset.
One of the most interesting result emerged from the analysis concenring KNN and SVM: theory suggests that SVM should have more predictive power and accuracy than KNN since it is far more complex. However, despite its simplicity KNN matches and even outperforms SVM. Moreover, this work confirms what other researches found out: CNN have the best performance and rgb features are very significant features even if they are easy to extract. To sustain the claim, the table 1 was built: it presents the best accuracy for each algorithm and for each feature.

Algorithms	1-NN	SVM	CNN
Grayscale	0.86	0.84	0.90
Rgr	0.91	0.92	0.94

Table 1: Best accuracy scores

It is also worth noticing that the accuracy rate achieved using rgb features are higher than all the accuracy rate achieved with grayscale features. However, these results seem to be 4% lower than the state-of-art in fruit classification. Indeed, the best accuracy rate reached in mixed fruit classification is 0.98 as reported in [2]. However, the comparison is not straightforward since the dimension of the dataset and the number of classes is different. Moreover, most of the literature focuses on one vs all classification while this work handles multiclass classification. Nevertheless, the objective of the project were fulfilled:

1. Grayscale features allows to achieve the remarkable accuracy (0.90).
2. Rgb features provide an 0.04 improvement of the previous result.
3. Despite its semplicity, knn has a remarkable accuracy rate (0.91).

6 Appendix: Code

```
library(jpeg)
library(imager)
library(class)
library(e1071)
library(FactoMineR)
library(factoextra)
library(keras)
library(tensorflow)

read_fruit_grayscale <- function(path,set,fruit, dim = 50){
  filenames <- list.files(paste(path,set,fruit,sep=""), pattern = "*.jpg")
  images_grayscale = list()
  i = 1
  for (name in  filenames){
    path_name = paste(path,set,fruit,"/",name,sep="")
    img <- grayscale(load.image(path_name))
    inmg <- resize(img,dim,dim)
    images_grayscale[[i]] = as.vector(img)
    i = i+1
  }
  return(images_grayscale)
}


#number of pixels to be used
dim = 50
path = "../fruits-360_dataset/fruits-360/"

# grayscale extraction
set = "Training/"
training_features_matrix <- NULL
training_labels_vector <- NULL

fruits = c ("Apple Golden 1", "Apple Golden 2",
"Apple Red 2","Apple Red 1", "Banana", "Apricot", "Blueberry")
i = 0
for (fruit in fruits){
  images_grayscale <- read_fruit_grayscale(path,set,fruit)
  images_grayscale <- matrix(unlist(images_grayscale), ncol = dim*dim, byrow = TRUE)
  training_features_matrix <- rbind(training_features_matrix,images_grayscale)
  labels <-replicate(dim(images_grayscale)[1], i)
  training_labels_vector <- c(training_labels_vector,labels)
  i <- i +1
}
```

```
set = "Test/"
test_features_matrix <- NULL
test_labels_vector <- NULL
i = 0
for (fruit in fruits){
  images_grayscale <- read_fruit_grayscale(path,set,fruit)
  images_grayscale <- matrix(unlist(images_grayscale), ncol = dim*dim, byrow = TRUE)
  test_features_matrix <- rbind(test_features_matrix,images_grayscale)
  labels <-replicate(dim(images_grayscale)[1], i)
  test_labels_vector <- c(test_labels_vector,labels)
  i <- i +1
}


#PCA
t1 <- Sys.time()
pca <- prcomp(training_features_matrix,center = TRUE, scale=TRUE)
t2 <- Sys.time()
pca_execution_time <- t2 - t1
# pca_execution_time = Time difference of 5.653888 mins
pr_var <- (pca$sdev[1:200])^2
prop_varex <- pr_var/sum(pr_var)
plot(cumsum(prop_varex), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained")


training_features_matrix_pc <- pca$x
test_features_matrix_pc <- predict(pca, newdata = test_features_matrix)

#BASELINE
neighbours <- c(1,3,5,7,9,11)
features <- c(3,10,50,100,150)
res_knn <- c()
time_knn <- c()
i <- 1
for (n_f in features){
  for (n in neighbours){
    t1 <- Sys.time()
    knn <-knn(training_features_matrix_pc[,1:n_f], test_features_matrix_pc[,1:n_f],
      as.factor(training_labels_vector), k=n)
    t2 <- Sys.time()
    time_knn[i] <- as.double(difftime(t2,t1,  units = "secs"))
    res_knn[i] <- sum(knn == test_labels_vector)/length(test_labels_vector)
    i <- i +1
```

```
  }
}

time_knn

res_knn

precision <- matrix(res_knn,ncol=length(features))
matplot(precision, type = c("b"),pch=1,  xlab = "Neighbours",
 ylab="Accuracy", main="KNN ACCUARCY", xaxt = "n")
axis(1, at=1:length(neighbours), labels=neighbours)
legend("left", legend = features, col=1:length(features), pch=1, title="Features")

time <- matrix(time_knn,ncol=length(features))
matplot(time, type = c("b"),pch=1, xaxt = "n", xlab = "Neighbours",
ylab="Execution time (s)", main="KNN EXECUTION TIME")
axis(1, at=1:length(neighbours), labels=neighbours)
legend("right", legend = features, col=1:length(features), pch=0.5, title="Features" ,cex = 0.7)

features <- c(3,10,50,100,150)
res_svm <- c()
time_svm <- c()
i <- 1
for (n_f in features){
  training_features_matrix_svm <- as.data.frame(training_features_matrix_pc[,1:n_f])
  training_features_matrix_svm$target <- as.factor(training_labels_vector)
  test_features_matrix_svm <- as.data.frame(test_features_matrix_pc[,1:n_f])

  t1 <- Sys.time()
  SVM <- svm(target ~ ., data = training_features_matrix_svm, kernel = "radial")
  t2 <- Sys.time()
  res <- predict(SVM, newdata = test_features_matrix_svm)
  time_svm[i] <- as.double(difftime(t2,t1,  units = "secs"))
  res_svm[i] <-     sum(res == test_labels_vector)/length(test_labels_vector)
  i <- i +1
}

time_svm
res_svm

plot(res_svm, type = c("b"),pch=1 , xlab = "Features",
ylab="Precision", main="SVM ACCURACY", xaxt = "n")
axis(1, at=1:length(features), labels=features)


plot(time_svm, type = c("b"),pch=1 , xlab = "Features",
```

```
ylab="Execution time (s)", main="SVM EXECUTION TIME", xaxt = "n")
axis(1, at=1:length(features), labels=features)


#CNN KERAS

y_train <- to_categorical(training_labels_vector, length(fruits))
y_test <- to_categorical(test_labels_vector, length(fruits))
x_train<-array_reshape(training_features_matrix,c(nrow(training_features_matrix),dim,dim,1))
x_test<-array_reshape(test_features_matrix,c(nrow(test_features_matrix),dim,dim,1))

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 16, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(50, 50, 1)) %>%
  layer_max_pooling_2d(pool_size = c(4,4)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(2, 2), activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(rate=0.5) %>%
  layer_dense(units = 300, activation = "relu") %>%
  layer_dense(units = 200, activation = "tanh") %>%
  layer_dense(units = length(fruits), activation = "softmax")

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 16, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(50, 50, 1)) %>%
  layer_max_pooling_2d(pool_size = c(4,4)) %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(4, 4)) %>%
  #layer_conv_2d(filters = 128, kernel_size = c(2, 2), activation = "tanh") %>%
  #layer_max_pooling_2d(pool_size = c(4, 4)) %>%
  #layer_conv_2d(filters = 128, kernel_size = c(2, 2), activation = "tanh") %>%
  layer_flatten() %>%
  layer_dropout(rate=0.4) %>%
  layer_dense(units = 300, activation = "relu") %>%
  layer_dense(units = 200, activation = "relu") %>%
  layer_dense(units = length(fruits), activation = "softmax")

model %>% compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics='accuracy')

t1 <- Sys.time()
history<- model %>% fit(
  x_train,y_train,
  epochs=3,
```

```
  batch_size=50)
t2 <- Sys.time()
cnn_execution_time <-  as.double(difftime(t2,t1,  units = "secs"))
model %>% evaluate(x_test, y_test)

#RGB ANALYSIS
read_fruit_rgb <- function(path,set,fruit, dim = 50){
  filenames <- list.files(paste(path,set,fruit,sep=""), pattern = "*.jpg")
  images_rgb= list()
  i = 1
  for (name in  filenames){
    path_name = paste(path,set,fruit,"/",name,sep="")
    img <- load.image(path_name)
    inmg <- resize(img,dim,dim)
    images_rgb[[i]] = as.vector(img)
    i = i+1
  }
  return(images_rgb)
}

#number of pixels to be used
dim = 50
path = "../fruits-360_dataset/fruits-360/"

# rgb extraction

set = "Training/"
training_labels_vector <- NULL
fruits = c ("Apple Golden 1", "Apple Golden 2", "Apple Red 2",
"Apple Red 1", "Banana", "Apricot", "Blueberry")

i = 0
training_features_matrix_rgb <- NULL
for (fruit in fruits){
  images_rgb <- read_fruit_rgb(path,set,fruit)
  images_rgb <- matrix(unlist(images_rgb), ncol = dim*dim*3, byrow = TRUE)
  labels <-replicate(dim(images_rgb)[1], i)
  training_labels_vector <- c(training_labels_vector,labels)
  training_features_matrix_rgb <- rbind(training_features_matrix_rgb,images_rgb)
  i <- i +1
}


set = "Test/"
test_labels_vector <- NULL
```

```
i = 0
test_features_matrix_rgb <- NULL
for (fruit in fruits){
  images_rgb <- read_fruit_rgb(path,set,fruit)
  images_rgb <- matrix(unlist(images_rgb), ncol = dim*dim*3, byrow = TRUE)
  test_features_matrix_rgb <- rbind(test_features_matrix_rgb,images_rgb)
  labels <-replicate(dim(images_rgb)[1], i)
  test_labels_vector <- c(test_labels_vector,labels)
  i <- i +1
}

#PCA
t1 <- Sys.time()
pca_rgb <- prcomp(training_features_matrix_rgb,center = TRUE, scale=TRUE)
t2 <- Sys.time()
pca_rgb_execution_time <- as.double(difftime(t2,t1,  units = "min")) # 57.14827 minutes
pr_var_rgb <- (pca_rgb$sdev[1:200])^2
prop_varex_rgb <- pr_var_rgb/sum(pr_var_rgb)
plot(cumsum(prop_varex_rgb), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained")

training_features_matrix_rgb_pc <- pca_rgb$x
test_features_matrix_rgb_pc <- predict(pca_rgb, newdata = test_features_matrix_rgb)

#KNN
neighbours <- c(1,3,5,7,9,11)
features <- c(3,10,50,100,150)
res_knn <- c()
time_knn <- c()
i <- 1
for (n_f in features){
  for (n in neighbours){
    t1 <- Sys.time()
    knn <-knn(training_features_matrix_rgb_pc[,1:n_f], test_features_matrix_rgb_pc[,1:n_f], as.facto
    t2 <- Sys.time()
    time_knn[i] <- as.double(difftime(t2,t1,  units = "secs"))
    res_knn[i] <- sum(knn == test_labels_vector)/length(test_labels_vector)
    i <- i +1
  }
}

time_knn
res_knn


precision <- matrix(res_knn,ncol=length(features))
```

```
matplot(precision, type = c("b"),pch=1,  xlab = "Neighbours",
ylab="Accuracy", main="KNN ACCUARCY", xaxt = "n")
axis(1, at=1:length(neighbours), labels=neighbours)
legend("left", legend = features, col=1:length(features), pch=1, title="Features")

time <- matrix(time_knn,ncol=length(features))
matplot(time, type = c("b"),pch=1, xaxt = "n", xlab = "Neighbours",
ylab="Execution time (s)", main="KNN EXECUTION TIME")
axis(1, at=1:length(neighbours), labels=neighbours)
legend("right", legend = features, col=1:length(features), pch=0.5, title="Features" ,cex = 0.65)

features <- c(3,10,50,100,150)
res_svm <- c()
time_svm <- c()
i <- 1
for (n_f in features){
  training_features_matrix_svm <- as.data.frame(training_features_matrix_rgb_pc[,1:n_f])
  training_features_matrix_svm$target <- as.factor(training_labels_vector)
  test_features_matrix_svm <- as.data.frame(test_features_matrix_rgb_pc[,1:n_f])

  t1 <- Sys.time()
  SVM <- svm(target ~ ., data = training_features_matrix_svm, kernel = "radial")
  t2 <- Sys.time()
  res <- predict(SVM, newdata = test_features_matrix_svm)
  time_svm[i] <- as.double(difftime(t2,t1,  units = "secs"))
  res_svm[i] <-     sum(res == test_labels_vector)/length(test_labels_vector)
  i <- i +1
}

time_svm
res_svm


plot(res_svm, type = c("b"),pch=1 , xlab = "Features",
ylab="Precision", main="SVM ACCURACY", xaxt = "n")
axis(1, at=1:length(features), labels=features)
legend("right", legend = features, col=1:length(features), pch=1, title="Features")

plot(time_svm, type = c("b"),pch=1 , xlab = "Features",
ylab="Execution time (s)", main="SVM EXECUTION TIME", xaxt = "n")
axis(1, at=1:length(features), labels=features)

features <- c(3,10,50,100,150)
res_svm <- c()
time_svm <- c()
i <- 1
```

```
for (n_f in features){
  training_features_matrix_svm <- as.data.frame(training_features_matrix_rgb_pc[,1:n_f])
  training_features_matrix_svm$target <- as.factor(training_labels_vector)
  test_features_matrix_svm <- as.data.frame(test_features_matrix_rgb_pc[,1:n_f])

  t1 <- Sys.time()
  SVM <- svm(target ~ ., data = training_features_matrix_svm, kernel = "radial")
  t2 <- Sys.time()
  res <- predict(SVM, newdata = test_features_matrix_svm)
  time_svm[i] <- as.double(difftime(t2,t1,  units = "secs"))
  res_svm[i] <-     sum(res == test_labels_vector)/length(test_labels_vector)
  i <- i +1
}

time_svm
#[1] 11.048466  7.334074 20.145182 42.065335 66.613935 radial kernel
res_svm
#0.6754386 0.8802632 0.9164474 0.9195175 0.9151316 radial kernel

plot(res_svm, type = c("b"),pch=1 , xlab = "Features", ylab="Precision", main="SVM ACCURACY", xaxt =
axis(1, at=1:length(features), labels=features)

plot(time_svm, type = c("b"),pch=1 , xlab = "Features", ylab="Execution time (s)", main="SVM EXECUTI
axis(1, at=1:length(features), labels=features)



y_train <- to_categorical(training_labels_vector, length(fruits))
y_test <- to_categorical(test_labels_vector, length(fruits))
x_train<-array(training_features_matrix_rgb,c(nrow(training_features_matrix_rgb),dim,dim,3))
x_test<-array(test_features_matrix_rgb,c(nrow(test_features_matrix_rgb),dim,dim,3))

model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 16, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(50, 50, 3)) %>%
  layer_max_pooling_2d(pool_size = c(4,4)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(2, 2), activation = "relu") %>%
  layer_flatten() %>%
  layer_dropout(rate=0.5) %>%
  layer_dense(units = 300, activation = "relu") %>%
  layer_dense(units = 200, activation = "tanh") %>%
  layer_dense(units = length(fruits), activation = "softmax")

model %>% compile(
  optimizer='adam',
  loss='categorical_crossentropy',
```

```
  metrics='accuracy')

t1 <- Sys.time()
history<- model %>% fit(
  x_train,y_train,
  epochs=3,
  batch_size=50)
t2 <- Sys.time()
cnn_execution_time <-  as.double(difftime(t2,t1,  units = "secs"))
model %>% evaluate(x_test, y_test)
```

References

[1] *Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.*

[2] *Khurram Hameed, Douglas Chai, Alexander Rassau, A comprehensive review of fruit and vegetable classification techniques. Image and Vision Computing 80 (2018) 22-44.*

[3] *Samuel Macedo, Givanio Melo, Judith Kelner, A comparative study of grayscale conversion techniques applied to SIFT descriptors, SBC Journal on Interactive Systems, volume 6, number 2, 2015.*

[4] *Sandeep Kumar, Zeeshan Khan, Anurag jain, A Review of Content Based Image Classification using Machine Learning Approach, International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970) Volume-2 Number-3 Issue-5 September-2012*

[5] *Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze, Introduction to Information Retrieval, Cambridge University Press. 2008.*

[6] *Shai Shalev-Shwartz, Shai Ben-David,Understanding Machine Learning: From Theory to Algorithms*

[7] *Trevor Hastie, Robert Tibshirani, Jerome Friedman, The Elements of Statistical Learning, Data Mining, Inference, and Prediction, 2nd edition*