Universidad
Carlos III de Madrid

Statistical Learning 16/17

Project Report

Group Members

Pablo Bordons Estrada

Sergio Gámez Ruiz de Olano

Pierre Mercatoris

Mohammadmehdi Fayazbakhsh

January 2017

1- Introduction:

The goal of this project is to recognise hand postures from grayscale photographs. There are over 4800 images with 6 different types of hand postures (see fig. 1), from 10 different persons. The variables of each image will either be pixels or area of the image with its grayscale intensity. The data will be downloaded from S. Marcel collection of images.

The idea came to us as the analysis of images implies very high dimensionality of data and will require dimension reduction. As none of us has been working with the analysis of images and we all wish to understand better how to extract information from other kinds of data, this dataset/analysis seemed best suited to this project and our curiosity.

To achieve this, we are going to apply data manipulation techniques such as cropping/scaling, normalisation and vectorisation along with dimensionality reduction (PCA and image gradient filter). Following this supervised statistical learning methods such as logistic regression, KNN, random forest and neural network will allow us to classify those images.

Figure 1: Six different hand postures contained in the grayscale images dataset.

2- Data Overview

This dataset consists of 4800 hand pictures, with 6 different hand gestures. The images have different observations depending on the image type, as we can see:

Gesture	**A**	**B**	**C**	**Five**	**Point**	**V**
N. Obs	1329	487	572	654	1395	435

Both the A and Point gesture have around two times more observations than the rest image classes. That might be an issue, as having unbalanced classes can be challenging when building the classifier.

The images are portable pixmap (.ppm) format images, that format has no compression applied on the images, which result on a heavier dataset, but also, in adding no noise coming from the compression process.

The images have these resolutions: 84x72, 82x70, 76x66 and 64x56. There is an issue, as it would be convenient to have a fixed resolution for every image. In order to fix this issue we considered both cropping, and scaling.

Each image, has 3 channels: Red, Green and Blue. Here we can see an example of an "A" image:

Figure 2: RGB image example.

That is the image, resulting of combining the 3 image channels. This is the 3 channels that compose that sample:

Red Channel Green Channel Blue channel

Figure 3: RGB channels from image example

The previous sample has a 76x66 image resolution, which mean that each channel have 76x66 pixels (5016 pixels). That means, that for this image we have a 76x66x3 dimensionality, around 15 thousand pixels. Taking into account that we have only 4800 images, if we wanted to build a model using the raw data, we would need to work on dimension reduction techniques, or image transformation to generate more synthetic samples.

In our case, we will work with the grayscale image version, result of combining each channel as follows:

grey_image=0.299*image@red + 0.587*image@green + 0.114*image@blue

We generate the grey image combining linearly the red, blue and green channels. The linear coefficients are related to the average sensitivity of our eyes for each color spectrum.

The grayscale image of the previous sample looks like this:

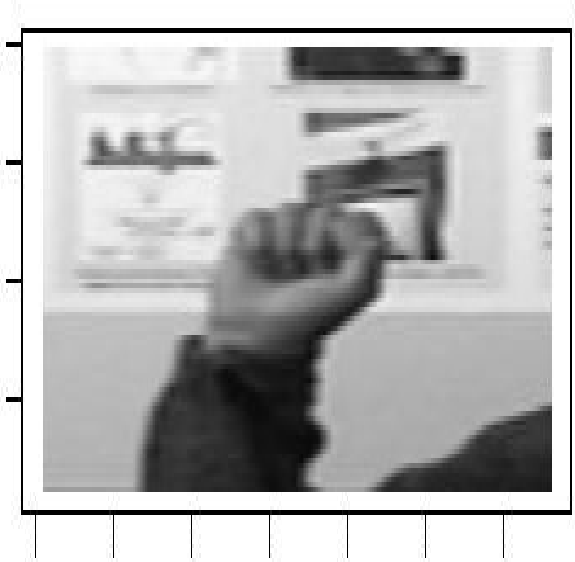

Figure 4: Grayscale image

As stated before, we have 6 image classes, this is an example of each class:

A class

B class

C class

Five class Point class V class

Figure 5: Six different hand postures classes contained in the dataset

3- Image preprocessing

3.1- Cropping/resizing

In order to have all the images with the same resolution, we needed to first resize the images, as we had a different range of sizes. This is the first step to prepare the image to be modelized.

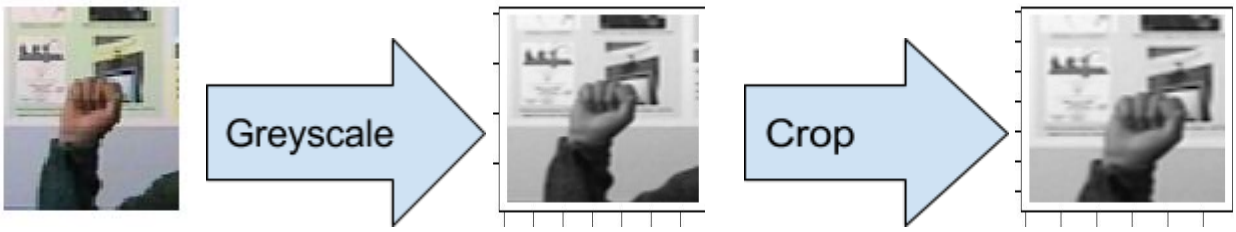

Figure 6: Image cropping manipulation workflow

As we can see, in the cropped image, the borders of the image are deleted, in order to fit a global resolution of 64x56. In general, while doing this cropping we center the image, and delete background, that in general might be a noise source. However, if the hand is not centered or the image has a much higher resolution, some important information could be lost. This is the reason why we have decided to proceed with scaling.

Scaling allows us to be certain that no relevant data is lost, but will also increase the importance of the background in further analysis. Additionally scaling also means reducing the resolution of hand.

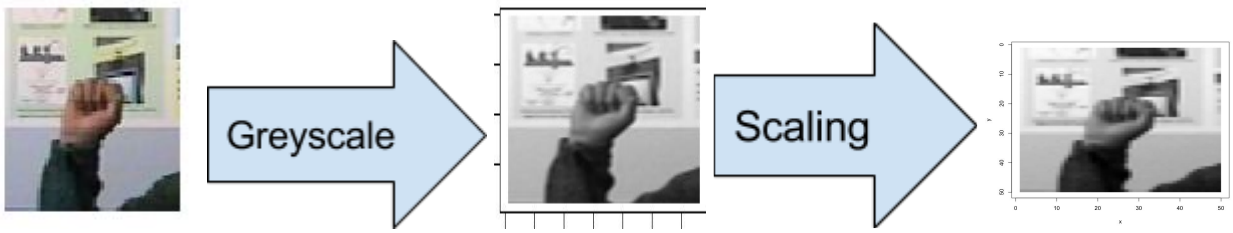

Figure 7: Image scaling manipulation workflow

3.2- Vectorization

So far, we have been working with images, 3 layer matrices (in the RGB case, 1 in a grayscale image). All machine learning models we work with, need to work with vectors, i.e. every sample must be a vector. In our scenario, we must transform every image grayscale matrix, to a vector. That means, that if every image is a 64x56 matrix we will get a vector of 3584 dimensions.

In order to do so, we extract the first row as a vector, and stack row after row in the vector. Doing this, we resolve the need to feed a vector to our model. In the other hand, we lose some information, as space correlations might be difficult to track in a vectorized image. Some advanced deep learning models, as convolutional neural networks use this spatial correlation to learn features from the image, building in different layers spatial filters that extract features looking at the patch, not just at a pixel level.

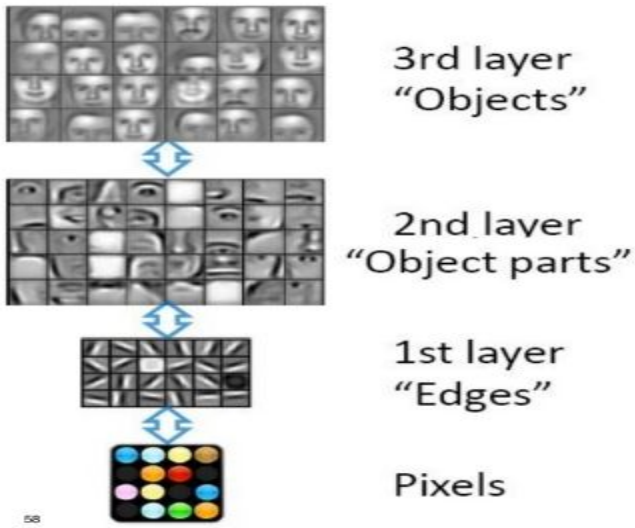

Figure 8: Feature representation of image in a neural network. Retrieved from https://image.slidesharecdn.com/20415dlbirdseyeview-150420223125-conversion-gate02/95/deep-learning-a-birds-eye-view-58-638.jpg?cb=1429569903

These state of the art models are interesting for image recognition, but they need a huge volume of observations. As in our case we only have 4800 samples to work with. We will see later whether this was sufficient for such statistical learning technique.

3.3 Normalization

During the normalization, we transform all image pixels from an arbitrary pixel intensity, to a fixed 0-1 pixel

intensity. This is the histogram of a sampled grey image:

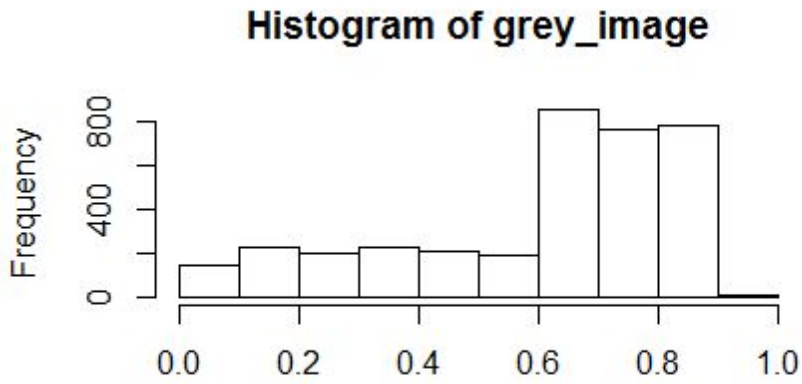

Figure 9: Histogram showing range of values in grey image.

The maximum pixel intensity of this image is 0.91, that means that during the normalization process, the pixel range intensity will be enlarged, from 0-0.91 to 0-1. This is the resulting histogram of the normalized image:

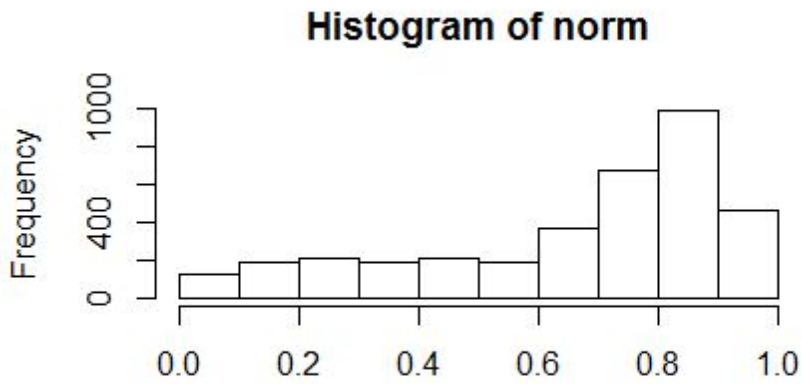

Figure 10: Histogram showing range of values in normalised grey image.

As we can see, now the image has pixel intensity has a range from 0 to 1. Below we can see the impact that the normalization has in the image perception:

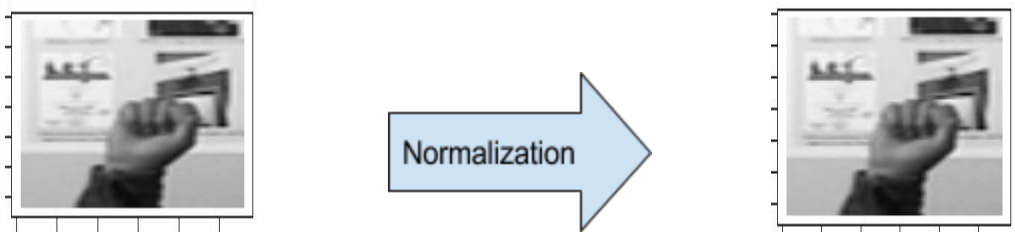

Figure 11: Effect of normalisation on image

Looking at the image, there is not a big difference from one to another, but that normalization is needed to have all images in the 0-1 range.

3.4- Gradient

A gradient filter, computing the change of intensity between pixels along y and x axis, was used in order to detect the edges of the features present in the images. This tricks permits to reduce greatly the amount of unnecessary information present in each image.

To do this, the function “imggradient()” from the R package “imager” was used. Below is an example of this applied filter.

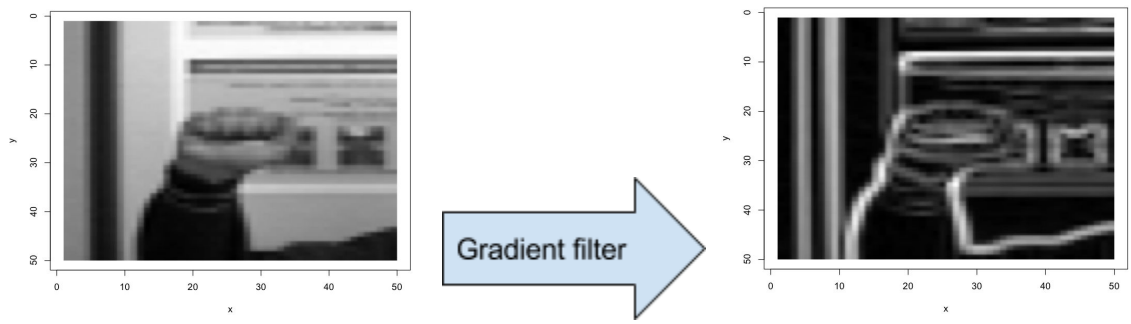

Figure 12: Effect of gradient filter on image

3.5- PCA

In our case, we use PCA to reduce the dimensionality of our data. PCA is able to find an alternative vector space, where the current image representation can be projected, while maintaining most of the information and highly reducing the dimensionality.

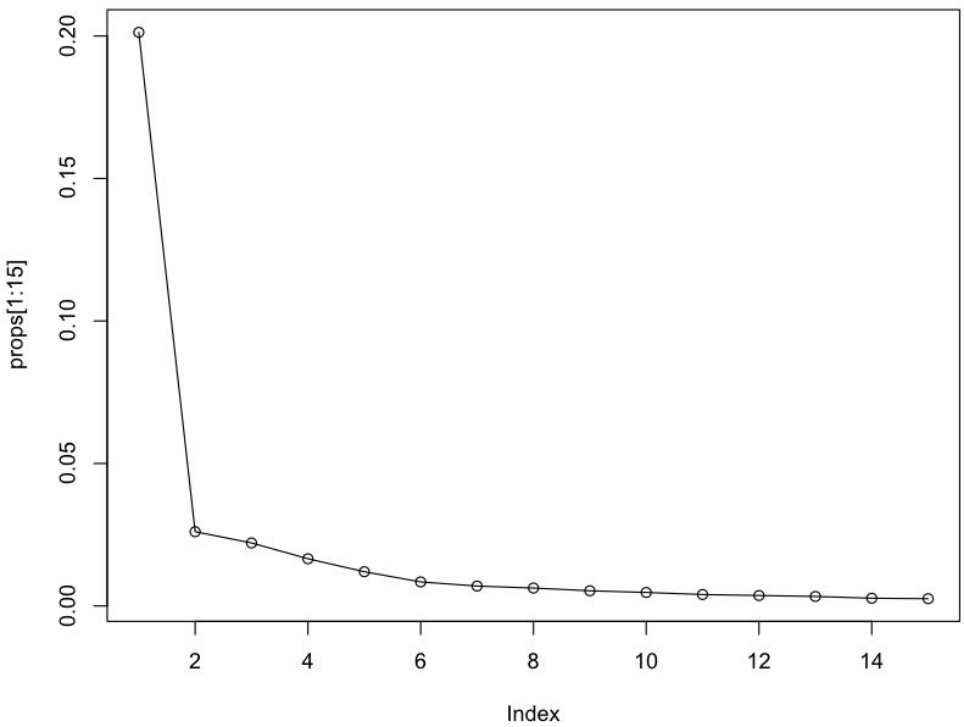

Figure 13: Proportion of variance explained by each principal component for images without gradient filter

In the previous image, we can see the proportion of the total variance explained by each principal components. We can see that the first 5 components cumulatively explain much more variance than the following 5 components. In our case, we use PCA to be able to feed the projected version of the images, in a reduced set of dimensions, so can avoid the curse of dimensionality.

The main preprocessing pipeline we use during this job is the following:

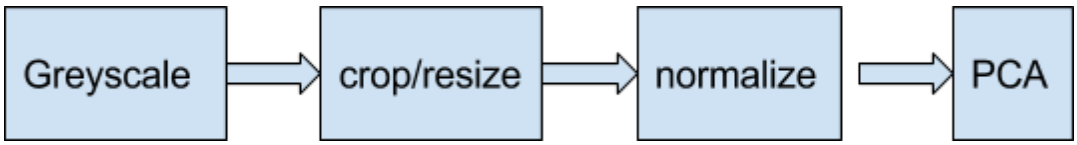

Figure 14: Workflow applied to images without gradient

That means, that approximately, we go from a 4800 observations dataset with 2500 dimensions, to approximately 5 after this pre-processing pipeline.

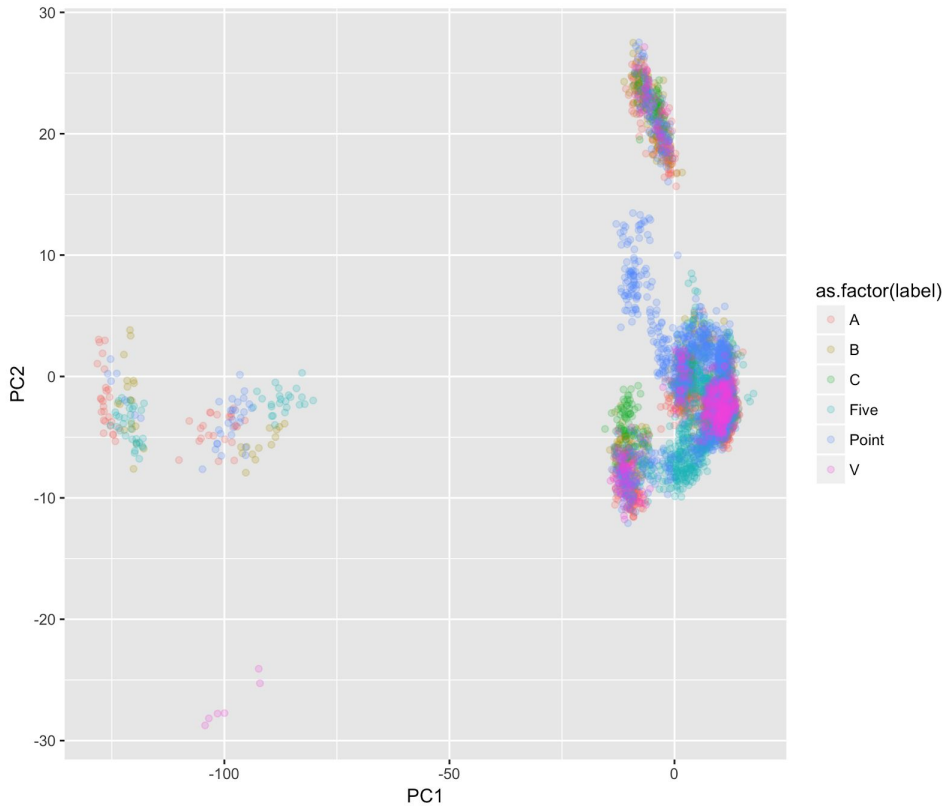

Figure 15: Scatter plot of all images from the 2 first principal component

The previous image is a visualization of the two principal components of our dataset. There is high class overlap, but using only two components there is already some samples that might be easily separable.

The following workflow was used to get the PCA of gradient image:

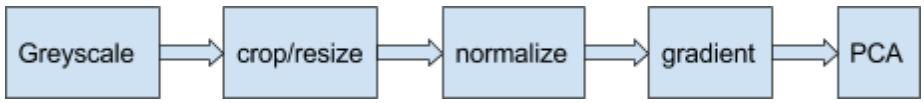

Figure 16: Workflow applied to images with gradient

The gradient filter is showing to be effective as here only 3 principal components are required to explain most of the variance.

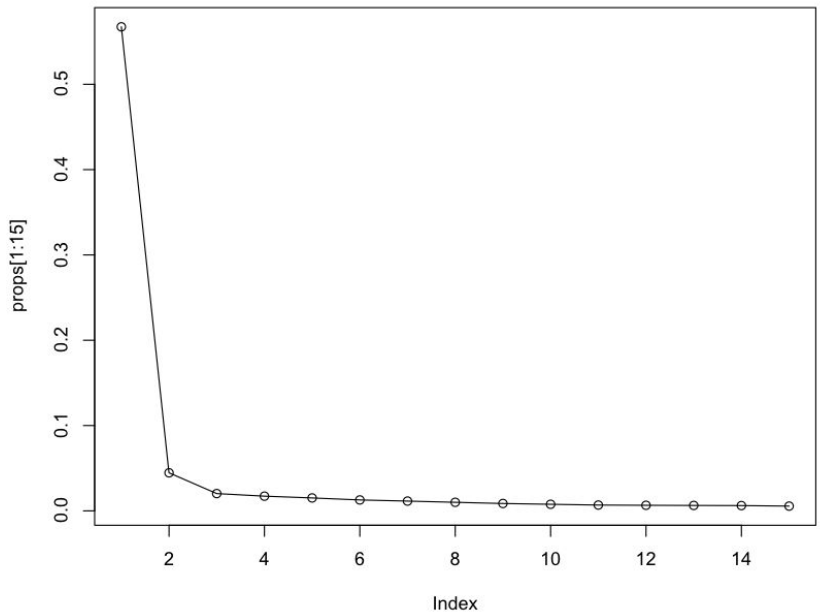

Figure 17: Proportion of variance explained by each principal component for images with gradient filter

The following image shows again the 2 first principal components from the gradient images. Some clustering is visible, but difficult to observe in 2 dimensions.

Figure 18: Scatter plot of all images from the 2 first principal component for images with gradient filter.

4- Modelling

4.1- Logistic regression

4.1.1- Description

Logic regression or Logistic regression is a modelling technique that can be used in classification problems involving categorical variables, often binary —the logit model gives the probability for a given instance to belong to a class (yes or not). It can be described as a generalized linear model (GML) where the linear function is replaced by the logit one.

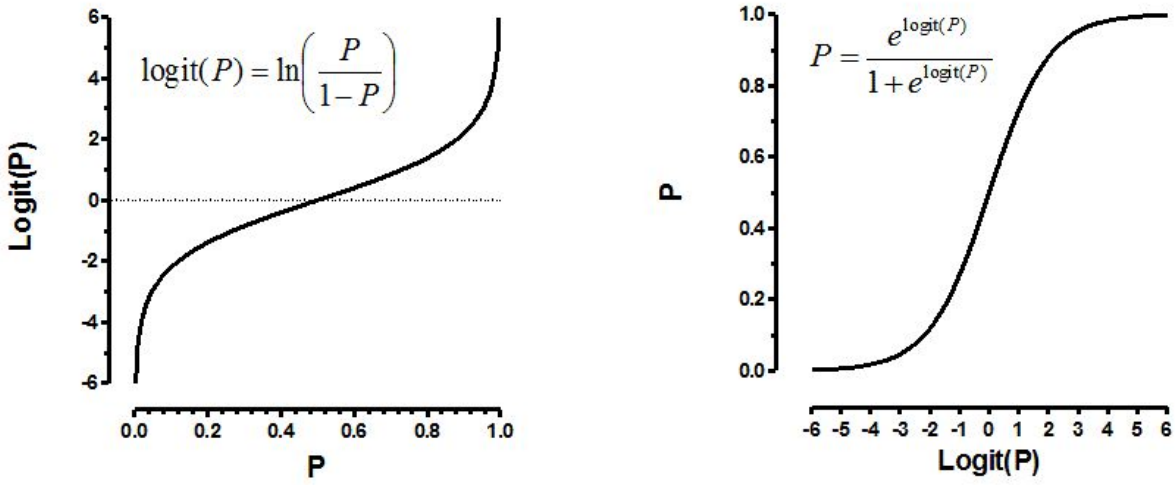

Figure 19: The logit function and its inverse, used to model the probability of an instance to belong to a class.

As it can be seen in the image, the function is very appropriate when trying to model a binary output as it is ranged from 0 to 1, converging to both extremes and, moreover, covering the middle values in a very narrow space of the domain.

In our case, and because there are 6 groups of images, it is necessary to adapt the approach for how the model is used. In short, we need to predict, one by one and for every class, the probability for an instance to belong to each of them. We will end up with 6 probabilities for each image. The prediction will be the highest of all of them.

This approach means that we can predict an image to belong to a class having little probability to belong to that class, as long as the probability is the largest one. An alternative approach would be to predict the class (yes or no) for every image and for every class, but this is more problematic. First, because it will cause many images to remain unclassified, and second, because for an image that has high probabilities to belong to several classes it would be difficult to decide to which of them to classify it (without comparing the probabilities as we do in the first approach).

In our work we are following the first approach, which can lead to more false positives, but will predict a class for every image.

4.1.2- R procedure

The model is applied to both, gradient processed images and gray-scaled ones. The dataset is divided in training and testing sets and PCA is used. For the classification we have used the first 25 components as discussed below.

The first necessary step is to create 6 dummy binary variables that will indicate solely the belonging to a specific class. We will create then also 6 different models, each of them predicting one of the classes over the 25 components. After the six predictions are done we need to combine them back into a single label column, picking for every dummy variable the one with the highest value. We do not want the highest probability, but the class containing it.

	P(A)	P(B)	P(C)	P(Five)	P(Point)	P(V)	Label
Instance	0.53	0.67	0.12	0.21	0.08	0.89	V

Figure 20: Example to illustrate how the label is computed from the probabilities. Note: not real data.

After the final prediction has been done we can compare it with the real one in order to obtain an estimation of the model. In order to do that we simply calculate the mean of the difference vector between the both. We obtain an accuracy of 71.43%.

4.1.3- Discussion

In order to obtain more information about the model, it is interesting to see which classes have more accuracy.

	results.label						
testing$label	A	B	C	Five	Point	V	Row Total
A	100	3	2	1	23	3	132
	0.758	0.023	0.015	0.008	0.174	0.023	0.273
	0.676	0.064	0.039	0.016	0.160	0.094	
	0.207	0.006	0.004	0.002	0.048	0.006	
B	2	31	5	1	5	4	48
	0.042	0.646	0.104	0.021	0.104	0.083	0.099
	0.014	0.660	0.098	0.016	0.035	0.125	
	0.004	0.064	0.010	0.002	0.010	0.008	
C	2	7	43	0	5	0	57
	0.035	0.123	0.754	0.000	0.088	0.000	0.118
	0.014	0.149	0.843	0.000	0.035	0.000	
	0.004	0.014	0.089	0.000	0.010	0.000	
Five	5	0	0	54	5	1	65
	0.077	0.000	0.000	0.831	0.077	0.015	0.135
	0.034	0.000	0.000	0.885	0.035	0.031	
	0.010	0.000	0.000	0.112	0.010	0.002	
Point	32	2	1	5	96	3	139
	0.230	0.014	0.007	0.036	0.691	0.022	0.288
	0.216	0.043	0.020	0.082	0.667	0.094	
	0.066	0.004	0.002	0.010	0.199	0.006	
V	7	4	0	0	10	21	42
	0.167	0.095	0.000	0.000	0.238	0.500	0.087
	0.047	0.085	0.000	0.000	0.069	0.656	
	0.014	0.008	0.000	0.000	0.021	0.043	
Column Total	148	47	51	61	144	32	483
	0.306	0.097	0.106	0.126	0.298	0.066	

Figure 21: Accuracy table detailed for each of the 6 classes..

As we can see, the most successfully classified is the ‘Five’ image, and the one with lower score is the ‘V’ image. We can see, for ‘V’ images, with what other classes the algorithm is mistaking it. In the table we observe that ‘A’ and ‘Point’ are often taken as ‘V’.

Another interesting thing to observe is the probability distribution of each of the classes:

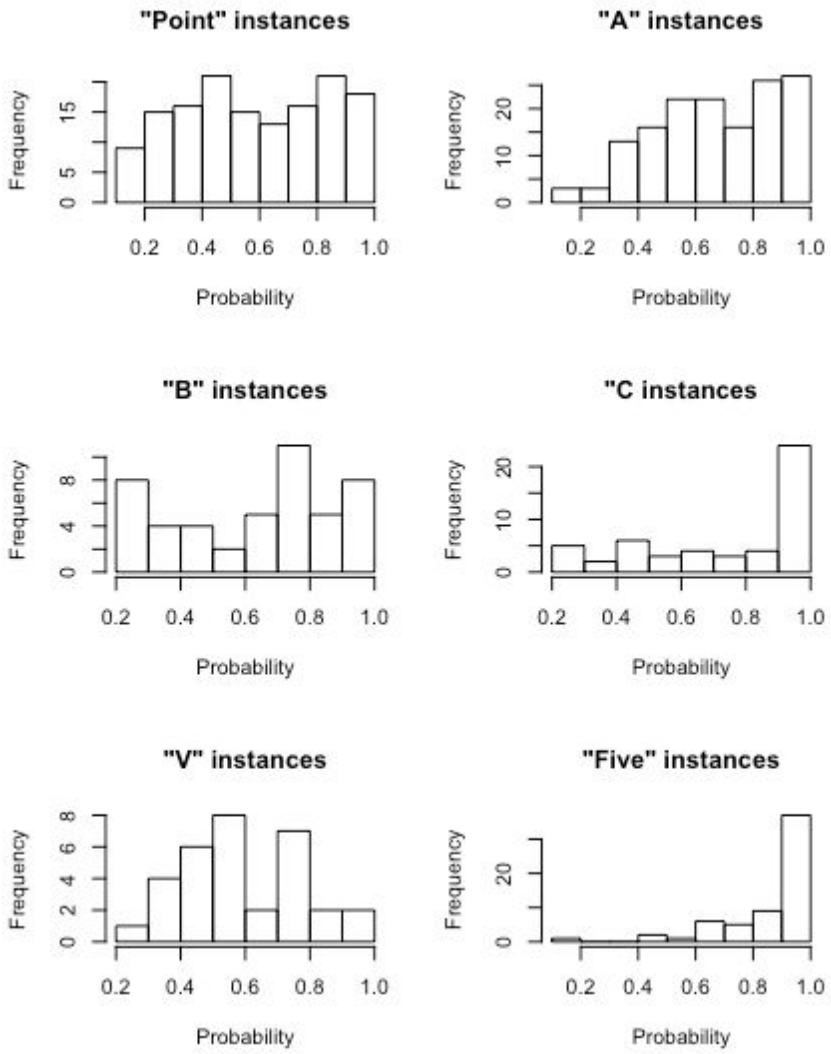

Figure 22: Probability distribution for each of the 6 classes.

As it can be seen in the different distributions, the model knows very clearly if an instance belongs to ‘C’ of ‘Five’ class, but for ‘Point’ and ‘A’ instances there histograms are flatter.

Overall we have obtained a very good result, taking into account the simplicity of the method.

4.2- k-nearest neighbors (kNN):

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems.

4.2.1- kNN Algorithm – Pros and Cons:

Pros: The algorithm is highly unbiased in nature and makes no prior assumption of the underlying data. Being simple and effective in nature, it is easy to implement and for these reasons has gained good popularity.

Cons: Indeed it is simple but kNN algorithm has been criticism for being extremely simple! If we take a deeper look, this does not create a model since there's no abstraction process involved. The training process is really fast as the data is stored verbatim (hence lazy learner) but the prediction time is pretty high with useful insights missing at times. Therefore, building this algorithm requires time to be invested in data preparation (especially treating the missing data and categorical features) to obtain a robust model.

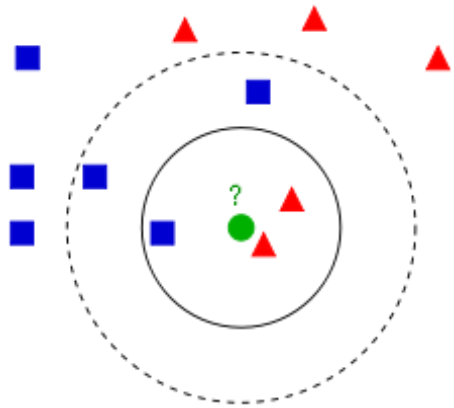

Figure 23: Classify new instances as the majority class of the k-nearest neighbours.

4.2.2- Steps for implementing kNN:

Step 1- Preparing and exploring the data

Normalizing numeric data: This feature is of paramount importance since the scale used for the values for each variable might be different. The best practice is to normalize the data and transform all the values to a common scale.

Here we have two data sets for kNN. One data set is with PCA without gradient and another is with PCA without gradient.

Everything in this step is already done in Image preprocessing part.

Step 2- Data collection

Creating training and test data set: The kNN algorithm is applied to the training data set and the results are verified on the test data set. For this, we would divide the data set into 2 portions in the ratio of 90:10 for the training and test data set respectively.

Step 3– Training a model on data and building the prediction model

The knn() function needs to be used to train a model for which we need to install a package 'class'. The

knn() function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number.
The kNN task can be broken down into writing 3 primary functions:

1. Calculate the distance between any two points

2. Find the nearest neighbours based on these pairwise distances

3. Majority vote on a class labels based on the nearest neighbour list

The steps in the following diagram provide a high-level overview of the tasks we'll need to accomplished.

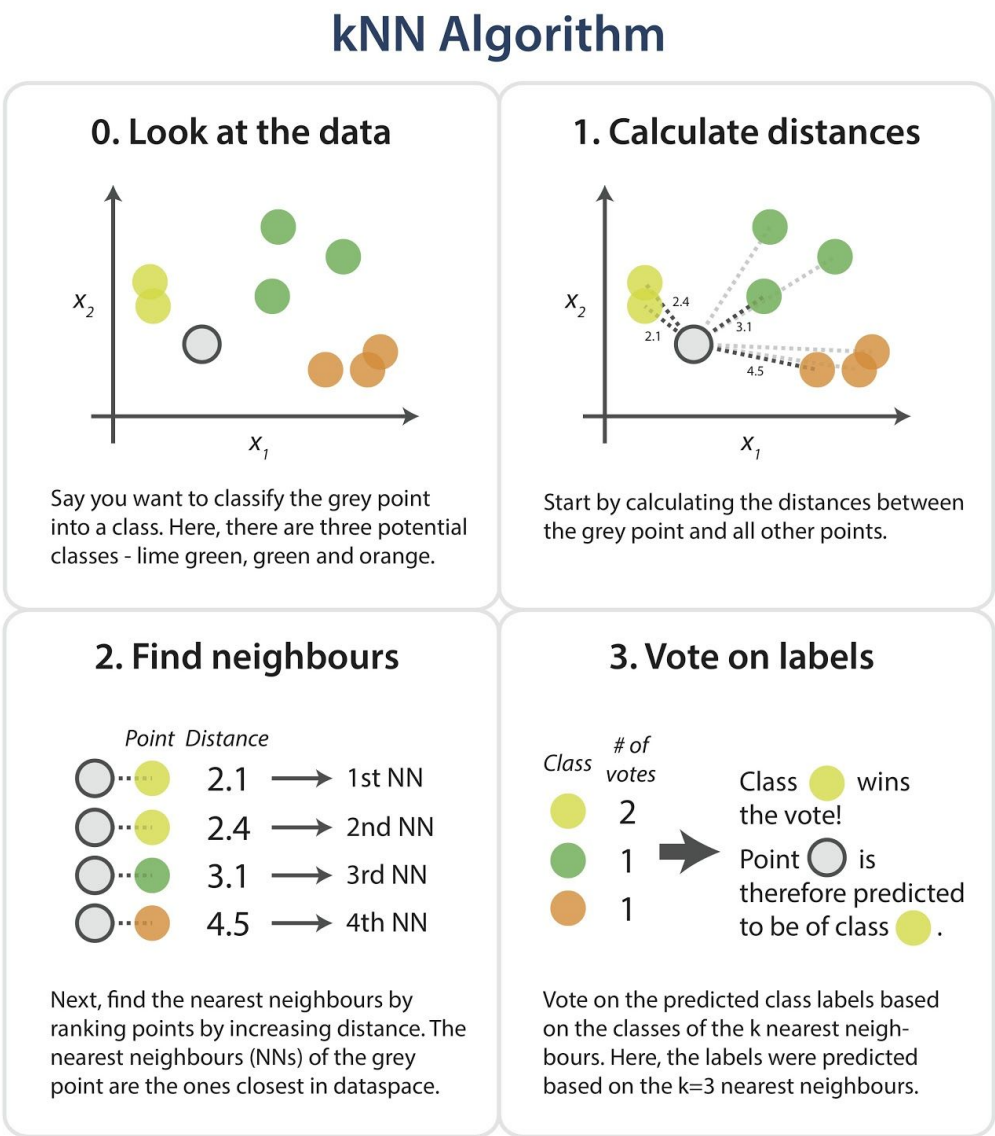

Figure 24: Overview of kNN algorithm steps

We can compare accuracy for different k in two methods (PCA with and without Gradient). In plots below it is clear that accuracies for different k in PCA with Gradient are higher than PCA without Gradient.

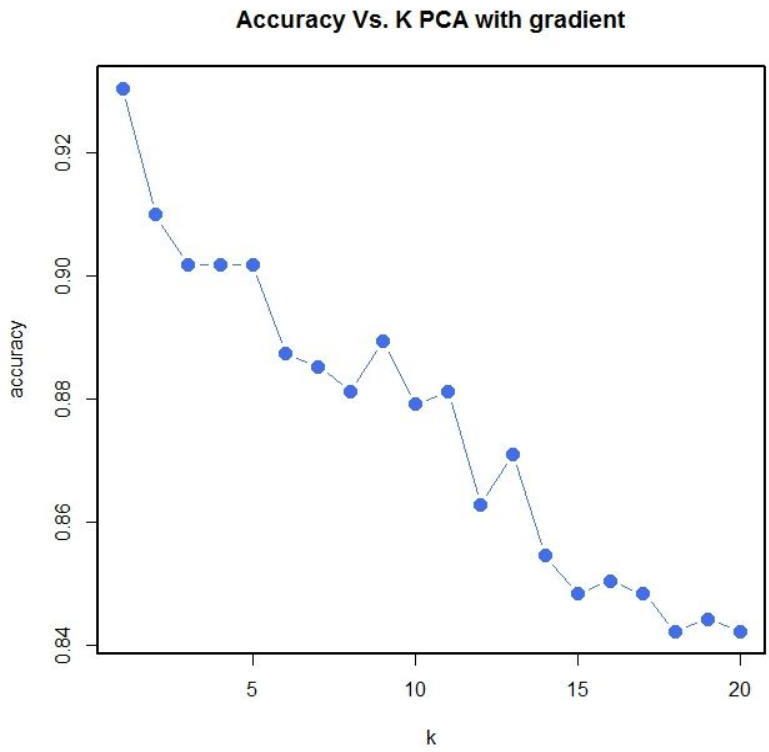

Figure 25: Accuracy for different k in PCA with Gradient.

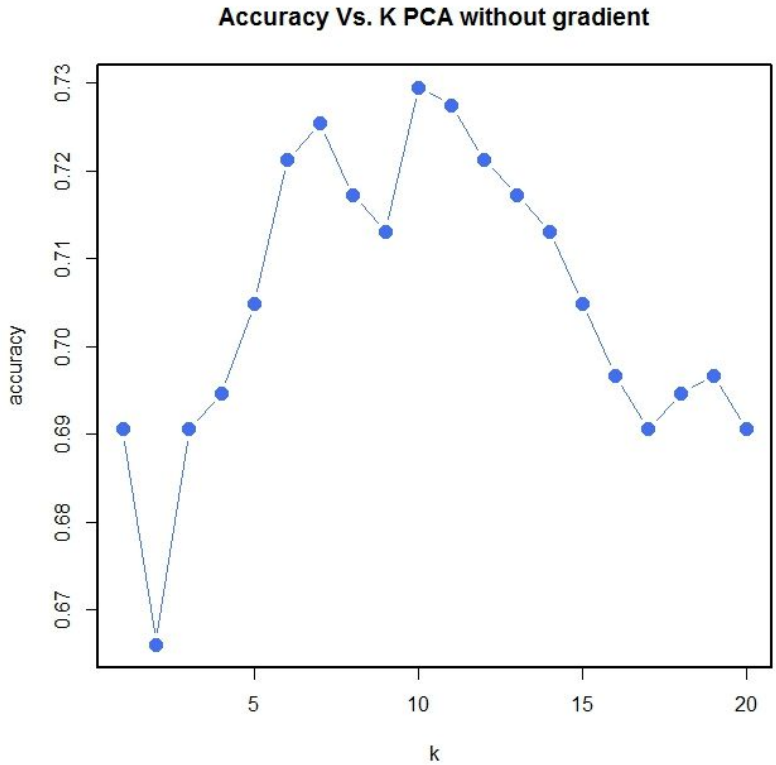

Figure 26: Accuracy for different k in PCA without Gradient.

Step 4– Evaluate the model performance

We have built the model but we also need to check the accuracy of the predicted values in knn_test_pred as to whether they match up with the known values in knn_test_labels. To ensure this, we need to use the

CrossTable() function available in the package ‘gmodels’.
First we show the table for PCA with Gradient and the accuracy of the model with k=5 in this method is 89.95 %. Then the table for PCA without Gradient which the accuracy in this method is 70.90 %. The results confirm that accuracy for PCA with Gradient is higher than accuracy for PCA without Gradient.

Table of accuracy for PCA with Gradient

	knn_test_pred						
knn_test_labels	A	B	C	Five	Point	V	Row Total
A	122	0	0	1	5	1	129
	0.946	0.000	0.000	0.008	0.039	0.008	0.264
	0.917	0.000	0.000	0.016	0.036	0.022	
	0.250	0.000	0.000	0.002	0.010	0.002	
B	1	43	0	1	0	1	46
	0.022	0.935	0.000	0.022	0.000	0.022	0.094
	0.008	0.915	0.000	0.016	0.000	0.022	
	0.002	0.088	0.000	0.002	0.000	0.002	
C	3	1	59	0	1	1	65
	0.046	0.015	0.908	0.000	0.015	0.015	0.133
	0.023	0.021	0.952	0.000	0.007	0.022	
	0.006	0.002	0.121	0.000	0.002	0.002	
Five	0	0	0	57	0	0	57
	0.000	0.000	0.000	1.000	0.000	0.000	0.117
	0.000	0.000	0.000	0.934	0.000	0.000	
	0.000	0.000	0.000	0.117	0.000	0.000	
Point	4	1	3	2	124	8	142
	0.028	0.007	0.021	0.014	0.873	0.056	0.291
	0.030	0.021	0.048	0.033	0.886	0.178	
	0.008	0.002	0.006	0.004	0.254	0.016	
V	3	2	0	0	10	34	49
	0.061	0.041	0.000	0.000	0.204	0.694	0.100
	0.023	0.043	0.000	0.000	0.071	0.756	
	0.006	0.004	0.000	0.000	0.020	0.070	
Column Total	133	47	62	61	140	45	488
	0.273	0.096	0.127	0.125	0.287	0.092	

Table of accuracy for PCA without Gradient

	knn_test_pred						
knn_test_labels	A	B	C	Five	Point	V	Row Total
A	104	3	4	0	13	6	130
	0.800	0.023	0.031	0.000	0.100	0.046	0.266
	0.671	0.073	0.073	0.000	0.094	0.182	
	0.213	0.006	0.008	0.000	0.027	0.012	
B	11	22	4	0	10	9	56
	0.196	0.393	0.071	0.000	0.179	0.161	0.115
	0.071	0.537	0.073	0.000	0.072	0.273	
	0.023	0.045	0.008	0.000	0.020	0.018	
C	8	3	39	0	5	2	57
	0.140	0.053	0.684	0.000	0.088	0.035	0.117
	0.052	0.073	0.709	0.000	0.036	0.061	
	0.016	0.006	0.080	0.000	0.010	0.004	
Five	0	3	0	63	3	0	69
	0.000	0.043	0.000	0.913	0.043	0.000	0.141
	0.000	0.073	0.000	0.955	0.022	0.000	
	0.000	0.006	0.000	0.129	0.006	0.000	
Point	20	6	4	3	103	1	137
	0.146	0.044	0.029	0.022	0.752	0.007	0.281
	0.129	0.146	0.073	0.045	0.746	0.030	
	0.041	0.012	0.008	0.006	0.211	0.002	
V	12	4	4	0	4	15	39
	0.308	0.103	0.103	0.000	0.103	0.385	0.080
	0.077	0.098	0.073	0.000	0.029	0.455	
	0.025	0.008	0.008	0.000	0.008	0.031	
Column Total	155	41	55	66	138	33	488
	0.318	0.084	0.113	0.135	0.283	0.068	

4.3- Random Forest

A Random Forest is an ensemble model, that combines the output of a set of trees. Each tree is trained with a random sample of both rows and columns. The final prediction is the average (or sum of votes) of all trees.

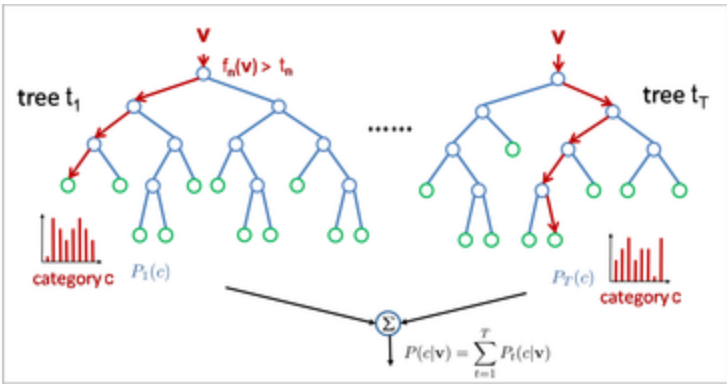

Figure 27: Structure of random forest algorithm

In this model, we picked the PCA representation of the images. This is an example of the feature importance of the model:

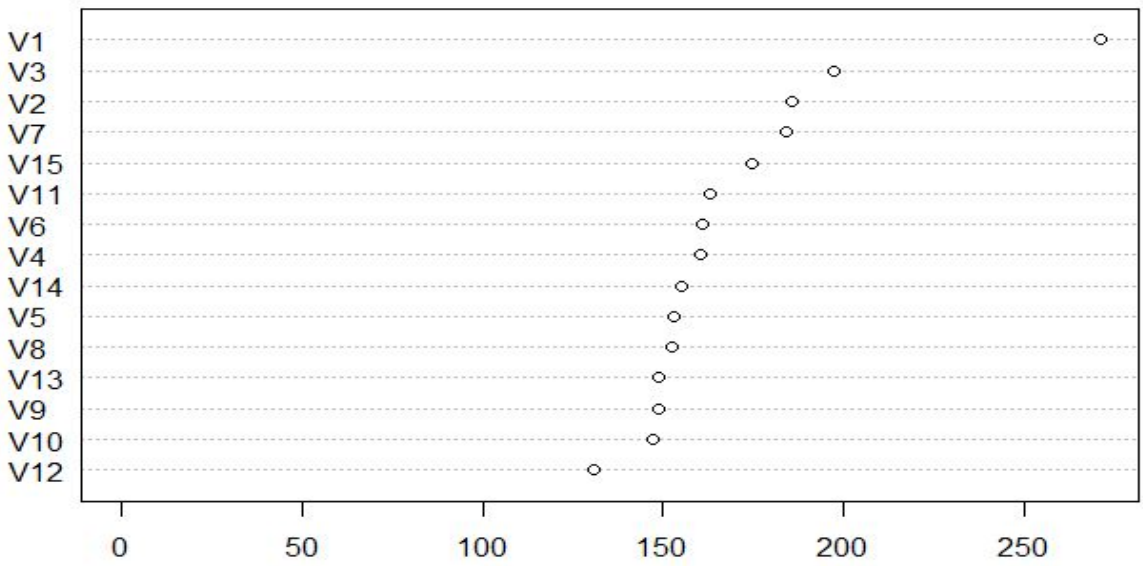

Figure 28: Importance of each component in the random forest algorithm.

We can see below the confusion matrix of this model used on cropped and no gradient pca:

	A	B	C	Point	V
A	257	7	0	18	3
B	8	81	3	7	3
C	8	1	88	8	3
Point	22	3	4	244	4
V	8	1	3	10	50

Figure 29: Confusion matrix of predicted vs labelled classes for cropped no gradient images

We can see that globally the confusion matrix main diagonal have a high volume of observations, meaning that the model performs well. Actually, this model has an accuracy of around 85%. This is a result, using a simple train-test split with a 80-20 random split, meaning that the train set has around 3374 observations, while the test set consists of 844 samples. In the previous example, we trained the model with 15 components. The following plot aims to assess the optimal number of components of the PCA, and the optimar node size hyperparameter of the Random Forest:

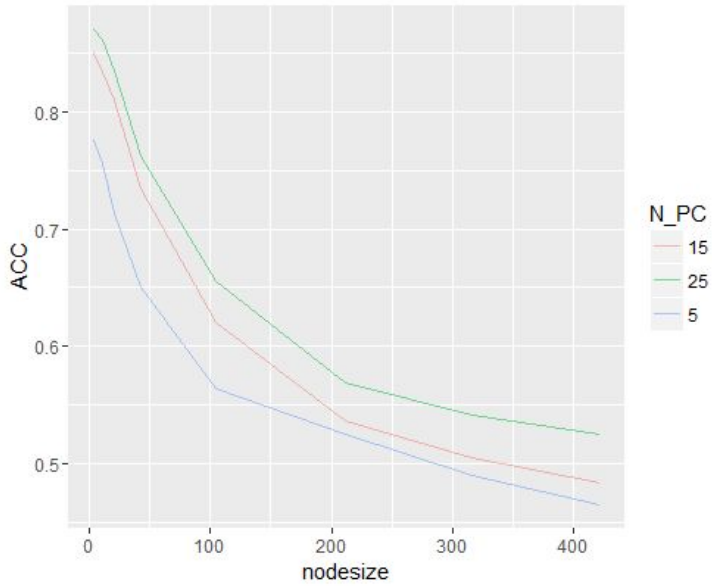

Figure 30: Hyperparameter tuning of random forest on cropped no gradient images

As we can see, the model performs better with 25 components, and a very low nodesize. It is worth mentioning that the gap from the 15 components performance is very similar to the 25 components one.

Below is a confusion matrix of this model used on scaled and no gradient pca:

```
> table(cv_test$label, cv_test$aux_pred)

         A   B   C Five Point   V
A      200   0   1   17    56   0
B       43   0   0    6    50   0
C       53   0   8   15    48   0
Five    10   0   0  100    20   0
Point   83   0   0   21   155   0
V       37   0   0    1    50   0
```

Figure 31: Confusion matrix of predicted vs labelled classes for scaled no gradient images

It seems that scaling has impacted the performance on B, C and V classes. This may be due to modifying the shape of the hands by scaling to a squared image, as the original images may originally be rectangular. For this reason the general accuracy of the model is lower than the one observed for cropped images.

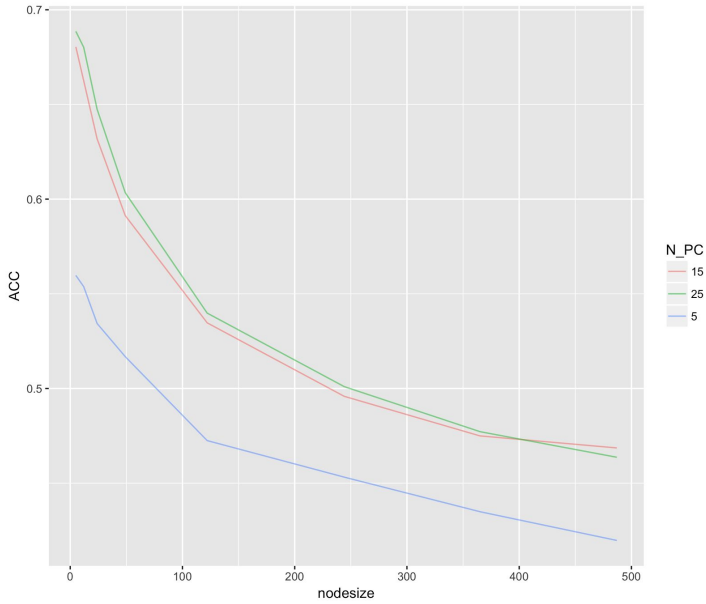

Figure 32: Hyperparameter tuning of random forest on scaled no gradient images

The next results are using the pca from gradient images which shows the best results for random forest. Although the classes B, C and V are still the least successful, they perform better than using images without the gradient filter.

```
> table(cv_test$label, cv_test$aux_pred)

          A   B   C Five Point   V
A       216   0   3    7    32   0
B        39  16  15    3    47   0
C        31   0  52    0    33   0
Five      5   0   0   89    17   0
Point   104   0   3   37   153   0
V        34   0   5    0    31   0
```

Figure 33: Confusion matrix of predicted vs labelled classes for scaled with gradient images

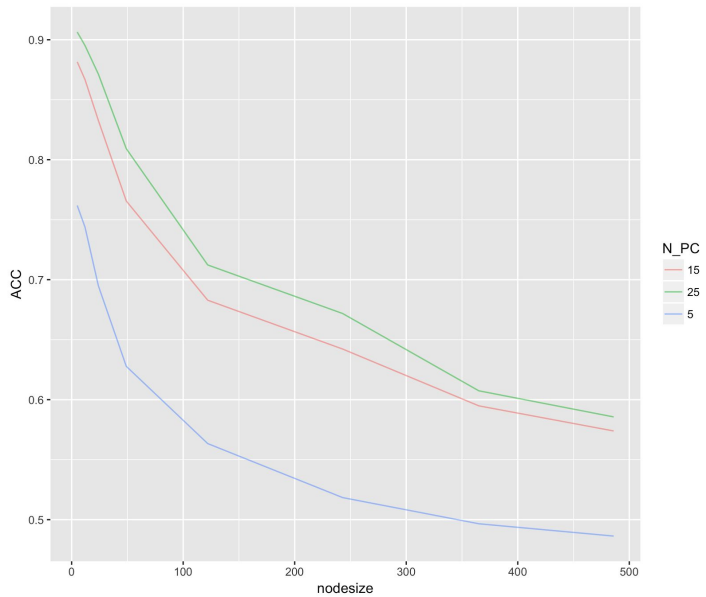

Figure 34: Hyperparameter tuning of random forest on scaled with gradient images

4.4- Neural Network

A neural network is an ensemble of neural units (neurons) interconnected across layers through various function (such as logistic or linear regressions). Each neuron can be connected to many other neurons and the functions allow to give more or less weights to data depending on the output class used. As more layers (as shown on the picture below) are added, the network can quickly become overwhelming and the inner workings considered as a black box.

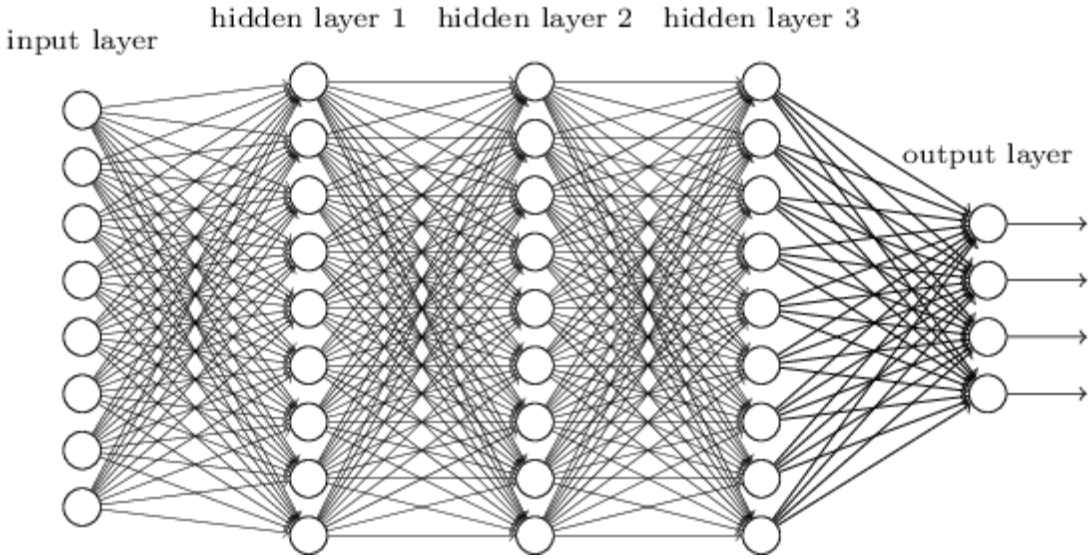

Figure 35: Neural network structure. Retrieved from http://neuralnetworksanddeeplearning.com/chap6.html

For this task, the data used was all the images along with their respective labels (A, Point, etc) resized to 28x28 pixels (too much memory required for computation on higher resolution) normalised grayscale images. Although the data used has been vectorised, parameters allow for the network to understand that each row represents a 2 dimensional image.

The network used here is a convolutional neural network (CNN), which is generally preferred for image recognition. In a CNN, each neuron of the input layer is a pixel of the image (here, 784 neurons). Using a 5x5 kernel, the image is scanned so as to allow for the detection across the whole image (as show a figure below).

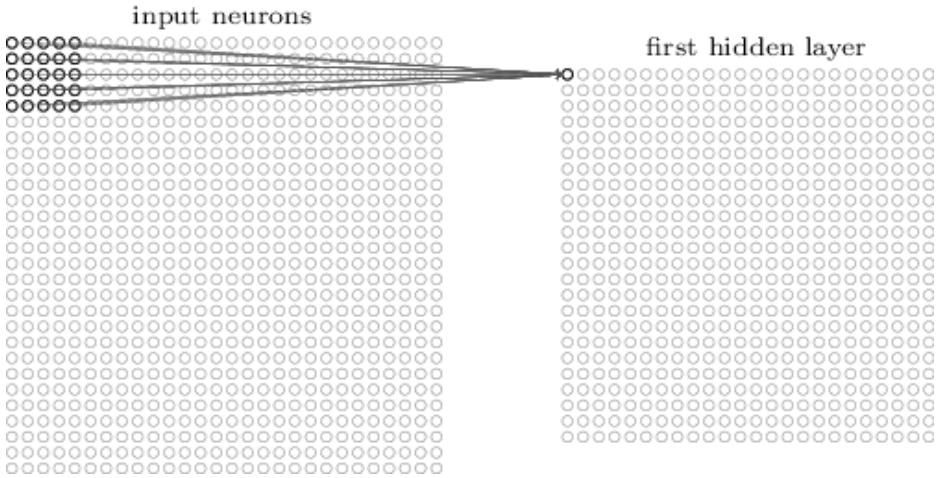

Figure 36: 5x5 kernel of CNN. Retrieved from http://neuralnetworksanddeeplearning.com/chap6.html

Using different weights on an activation function (in this case, hyperbolic tangent function) to detect for

various features. the input layer will result in a series of filters in the first hidden layer (now 24x24 neurons). The layers where this activation functions are used are named convolutional layers, and is the reason for the name of this type of neural network.

Finally, pooling layers allow to reduce summarise the output of the convolutional layers, by taking the maximum value of each 2 pixels, resulting in a matrix of 12x12 neurons. From there it is already possible to link to the wanted classes (as shown below with 10 classes).

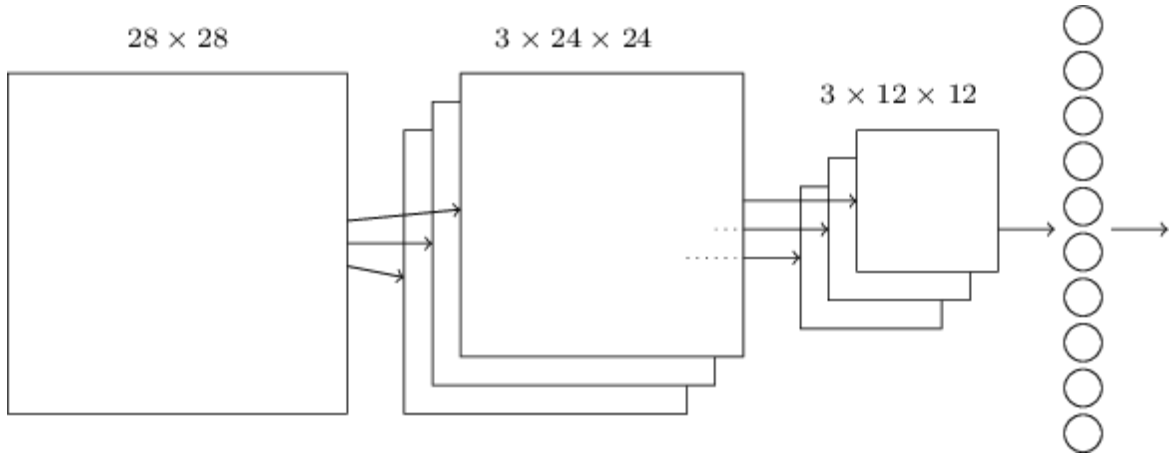

Figure 37: CNN structure. Retrieved from http://neuralnetworksanddeeplearning.com/chap6.html

4.4.1- R implementation and results

In this case, 2 series of convolutional (kernel of 5x5 with hyperbolic tangent activation function) and pooling layers (maximum of 2 pixels) were used followed by 2 series of fully connected layers (standard neural network).

This network as implemented using the R package "MxNet". The data, gray scale 28x28 images (normalised to 0-1 range) was split into a training (90%) and testing (10%) set. No crossvalidation was used as the learning phase is already computationally intensive.

```
> table(test[, 1], predicted_labels)
      predicted_labels
         1   2   3   4   5   6
  A    129   1   0   2   0   0
  B      1  43   0   2   2   0
  C      1   0  55   0   1   0
  Five   0   3   1  61   0   0
  Point  2   0   0   1 135   1
  V      2   0   0   0   3  38
> sum(diag(table(test[, 1], predicted_labels)))/sum(table(test[, 1], predicted_labels))
[1] 0.9524793
```

Figure 38: Confusion matrix of predicted vs labelled classes for scaled images (no gradient)

As seen in the image above, the network performed extremely well and had a mean accuracy of 95.24%. The most successfully recognised images were "A" with an accuracy of 97.72% and the worst accuracy (88.37%) was observed for "V".

5- Conclusion

Overall the data manipulation used (cropping/scaling, normalisation and vectorisation) along with dimensionality reduction (PCA and image gradient filter) allowed us to successfully use standard techniques of supervised learning. Each method has performed with an accuracy greater than 50%, demonstrating that each technique has worked.

The gradient filter proved to be a significant increase in classification performance and a great complement to PCA. As expected, the more complex the algorithm, the more accurate the result, as the performance increased from logistic regression, KNN and random forest up to the neural network. The higher performer was also the more computationally intensive as the analysis was only possible on an image set of reduced quality.

This work has allowed to show that standard can be used relatively easily to images and has granted the whole team with unvaluable experience and confidence to analyse images.

References

Data reference:

Marcel, Sebastien (1999). *Hand posture recognition in a body-face centered space.* Retrieved from http://www.idiap.ch/resource/gestures/

Resources:

Nielsen, Michael (Jan 2017) *Deep learning.* Retrieved from http://neuralnetworksanddeeplearning.com/

Mic (Aug 2016). *Image recognition tutorial in R using deep convolutional neural networks (MXNet package).*
Retrieved from:
https://www.r-bloggers.com/image-recognition-tutorial-in-r-using-deep-convolutional-neural-networks-mxnet-package/

Geitgey, Adam (Jun 2016). *Machine Learning is Fun! Part 3: Deep Learning and Convolutional Neural Networks.*
Retrieved from:
https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721#.uhwa5j80j

Appendix

All code used through this project is now publicly available at the following Github repository:
https://github.com/P-Mercator/Statistical_Learning_Project