Machine Learning

Image Gender Classification

using K Nearest Neighbor, Support Vector Machines and Convolutional Neural Networks methods

Laura Cebollero Carles Garriga Balbina Virgili

June 20, 2018

1 Introduction

During this course of Machine Learning we have seen a handful of methods to classify data. Until now, all of our work has been applied onto numerical data derived from metrics, aggregations and such.

However, these methods picked our interest on how they could be applied to images, which, at first glance for us, we did not think that it could have been converted into a numerical challenge.

Since we do not have any experience on having worked with image classification and only a little on just classification, we have decided to develop a binary image classification problem. More specifically, we are going to perform a supervised machine learning process consisting on classifying images containing faces from different people and classify them by biological gender: either female or male.

2 Goals and scope

Our **goals** are many and diverse:

1. Study how features from an image can be extracted for a posterior usage.
2. Apply different classifier methods and compare them.
3. Achieve at least 60% accuracy to demonstrate that the chosen classifier is better than tossing a coin (50%/50%).

Because of these goals, our work has a clearly defined **scope** which is:

1. Analise and perform the preprocessing needed for the chosen images.
2. Choose an accurate already-implemented feature extraction method that is useful for image **binary** classification, understand and apply it on our dataset.
3. Apply three different classification methods (KNN, SVM and CNN) to our extracted features.
4. Understand and tune the basic parameters of each classification method to be able to obtain the one that maximizes the results of our chosen dataset.
5. Compare the obtained results between different methods and choose the final best model for this problem.

3 The dataset

Our chosen dataset is called *Faces in the wild* [6]. This dataset is free access and includes both, a list of funneled images of a wide range of distinct people and two labels files, each one containing the name of the images for each gender.

More precisely, the funneled dataset contains 13.233 images from 5.749 people and all the images have been rotated so the face is centered and the eyes are at the same position. *Note that some faces are sideways and not frontal.* Moreover, thanks to the two labels files, we can know the gender of the person that appears on each image, by matching its name with those in the files.

Figure 1: Male

Figure 2: Female

4 Preprocessing

Since our problem is about image classification, the preprocessing developed differs greatly of what we would do with numerical data. The different steps included in our implemented preprocessing are the following ones stated.

4.1 Dataset composition

First of all, we downloaded the chosen dataset from Faces in the wild site [6] with the images funneled. After analyzing the composition of the downloaded files, we created a script in *bash* to simplify our reading of images, since they were grouped in folders by the name of who the person in the photo was and we decided to extract and locate each of them to the same folder.

4.2 Gender parity and diversity

Then, we realized that the dataset consisted of 2966 images of females and 10268 of males. So the input dataset is not balanced, as there are almost three times more males than females. Having such a clearly unbalanced dataset is not beneficial for us because, since roughly the 66% of the dataset is a male, then there would be predisposition for models to always select a male over a female. To avoid having this conflict and train both classes equally, **we have opted to force equality on the number of females and males.**

Additionally, we also faced that there were people in the dataset that had significantly more photos over others which only appeared once. For example, the most flagrant case we had was George W. Bush appearing up to 530 times with 530 different photos. As we do not want the gender to be biased on classifying someone according to the resemblance to the people that have the most number of appearances, **we have also chosen to allow only one photo per person.**

So, during our preprocessing, **first we forced to just use one photo per person and then we forced the gender parity**.

This diversity constraint, as well as, the gender parity have left us with the following images for our experiments:

Gender	**Initial**	**1st step**	**2nd step**
Female	2.965	2.073	2.073
Male	10.268	6.853	2.073
Total	**13.233**	**8.926**	**4.146**

And, finally, we needed to create a unique file with the labels of the chosen images for our experiment, and we preferred to have the dataset varied. For it, instead of just appending the information of one gender file after the other, we also mixed all the individuals' labels so there are no people of the same gender sequentially ordered in our dataset. This has been done to allow a manual splitting of the dataset or more efficient cross validation.

4.3 Image processing

As we want to classify people by his gender, the color of its skin is not relevant as the gender is not dependent of it. Also, we have noticed that each photo has a different luminous condition: some are redder while others are more bluish. So, because of these two phenomenons, we have chosen to work with the images in gray-scale only.

Moreover, we take advantage of the fact that the faces have been centered by the eyes but most images information contain the depiction of their background, which we are not interested in.

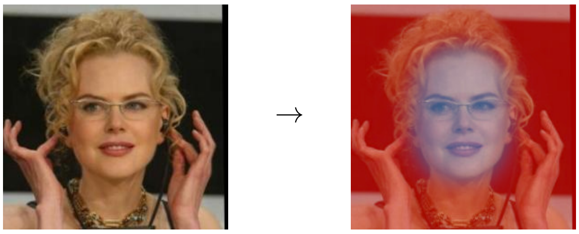


Figure 3: Intial picture vs. Relevant region

In blue we have the region of the face that we are interested on, whereas in red we have marked what we do not want, as the background does not determine which gender the person is. Also, we treat the hair of the person also as irrelevant because if a person does not have the hair visibly, we could mistake the background as the possible hair of the person in question.

So, **we have decided to crop each face automatically**, taking [80:180] on y and [80:170] on x, **to have only the features we are interested in, in gray-scale.**

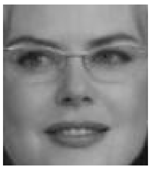


Figure 4: Female with face cropped in grayscale

4.4 Feature extraction: LBP

There are multiple methods to convert the image to numerical data that we can use for image classification. However, not all methods are adequate for this type of problem of gender classification.

After much research done, a first possible approach would be to compute the *eigenfaces* or *fisherfaces* [14]. However, we have have opted to use Local Binary Pattern[2], LBP in short, as our method to extract numerical information from the image.

This method works in grayscale and is not misled on obtaining the features depending on whether the lighting of the image is good or bad, which happens in EigenFaces and FisherFaces methods. LBP consists on taking the photo, cut it in different blocks, which would result in a grid. Then, cell per cell, we take the central pixel and with the help of the neighboring pixels, a LBP value is computed.

This value is obtained from comparing the central pixel with its neighboring ones. The neighborhood of a pixel is determined by a radius (how far from the central point are we going to check) and the number of points (how many samples) we are going to check.

So the radius and points dictate how big a block will be.

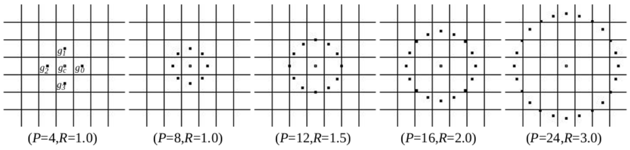


Figure 5: Differents points and radius

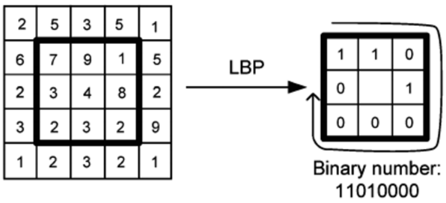
The comparison of the central point respect to the neighboring ones is as follows:

Let P be the neighboring points of a central point x.

$$f(x, p_i) = \begin{cases} 1, \text{ if } x <= p_i \\ 0, \text{ if } x > p_i \end{cases} \quad \forall p \in P$$

Or, in other words, if the intensity in a pixel in the neighborhood is greater (or equal) than the one of the central point, then we assign to that point a 1. Otherwise we assign a 0 to it.

This computation is done uniformly (without skipping any value) and in order starting from the top left corner of the neighborhood and then continued applying a clock-wise order:



2	5	3	5	1
6	7	9	1	5
2	3	4	8	2
3	2	3	2	9
1	2	3	2	1

1	1	0
0		1
0	0	0

Figure 6: LBP pattern reading

Now, in this example since there are 8 bits in there and in binary form, this means that we can obtain a value ranged from 0 to 255, which is the value that we will store. After we have gathered these values from all the blocks we obtain an histogram.

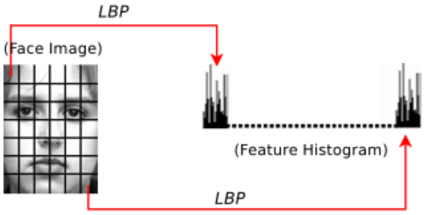


Figure 7: LBP applied to an image

So for each image we will have a feature histogram, which is what we will be using as the feature to classify our images in our methods.

4.4.1 Radius and points selection

Having seen how LBP works, we have now to choose the number of points we want to work with, as well as with which radius from the central pixel.

To take this decision we have decided to rely on two approaches:

1. Check previous studies on LBP applied to faces.
2. Create a controlled environment so we can compare results.

In the studies we have found and checked, we have seen that they orbit around a small number of points and a radius between 2 and 3[8]. In fact, there are studies [12] where they have applied some improvements in LBP where they only use 8 points with a radius of 2.

However, we are going to apply LBP uniformly as-is, or in other words, without complicated patterns, for improving LBP is not the focus on our study. It is merely a tool for us for feature extraction.

So having no clear reference on what is the best radius and number of points to have, we have complemented our findings with some experiments in a controlled environment.

This controlled environment consists on executing the K Nearest Neighbors algorithm with a fixed k, where $k = 7$.

The reason to use this method is due to Knn relying on the closeness/proximity between the individuals based on their values, since the distance is computed with them. We have fixed k to 7 as to see how well it works with a well defined locality environment.

So Knn right now is not used as a model to predict of what gender someone is, but to help us take the decision on which radius and number of points settle. **So these experiments are orientative.**

The training and test set have been the same in all cases (not randomized) to see how it behaved on each execution with different parameters. From the total dataset, the first 80% has been used as training and the last 20% to test.

Results with a fixed k = 7, expressed in %

R	P	Accuracy	Recall	Precision
1	8	60	53	62
2	**16**	**64**	**60**	**66**
3	**16**	**65**	**63**	**66**
3	**24**	**64**	**66**	**64**
4	16	62	53	65
4	24	63	54	66
8	24	61	55	62
8	48	61	55	62

In this table above we can clearly see that for a fixed $k = 7$, $P = 16 \vee 24$ and $R = 2 \vee 3$ are the preferred values, for their accuracy, recall and precision peaks.

We can see that the worst case is when locality is very strong, which is the case of when we are checking the immediate neighbor pixels with $P = 8$ and $R = 1$.

When $R \geq 4$ we can see that regardless of the number of points, locality is lost and thus accuracy also drops.

Since we had to choose between the three highlighted possibilities, we have seen that having a radius of 3 seems to work in our favor. And between 16 and 24 points, we have chosen to settle with 24 of them, so that we have as much information as possible in terms of the number of points.

So, to summarize, **our feature extraction will be done with LBP with the parameters $P = 24$ and $R = 3$.**

5 Classification methods applied

To classify the people from our dataset, we have taken three approaches that are very different between them as to see their keenness when applied to image classification. These three approaches have been:

- K Nearest Neighbors.
- Support Vector Machines.
- Convolutional Neural Networks.

All of them have been implemented in Python as well so their comparison in terms of code is easier.

In each of them, we are retrieving the accuracy and the logarithmic loss achieved with the model using different parameters. Thanks to the accuracy parameter we can see how well fitted our model is respect to the data and, the log. loss represents the empirical error resulted in our model. With it, we can see how well the model performs.

The accuracy is expressed in a percentage form meanwhile the Log. Loss ranges from 0 to ∞. In the first case, we want to **maximize the accuracy** as much as possible so that it classifies as perfectly as possible new individuals. In the second case, we want to **minimize the Logarithmic loss**, which means minimizing the empirical error found in our model.

These two resulting values will be used to compare the models trained with our data. This way, we will be able to choose the optimal parameters for each model as the ones that minimize the logarithmic loss on the execution.

5.1 K-Nearest Neighbors

The first method we have applied is K Nearest Neighbors, also known as KNN.

This method allows the computation of the distance between each individual in a matrix X with N individuals. Usually, this distance is computed with the Euclidean distance onto all the features of the individual, but other distance options are possible such as the Mahalanobis distance.

Then, for each new individual we want to compare to the existing ones, we compute the distance of that new individual to each one in the set of N. Then, the algorithm takes the K nearest to it to decide to which class he is closer to, by counting the number of occurrences on each one on those k nearest neighbors.

We have used the implementation of *Knn* from the *Scitkit Learn* [11] package in Python. This has allowed us to focus on the execution of Knn itself rather on implementing it. The distance method used in this implementation by default is the Euclidean one.

5.2 Methodology on KNN

The methodology applied on Knn consists on **performing a 10 fold cross validation on our whole data for different k.**

The selected k are all odd, so that no ties are allowed when counting the genders on the k nearest neighbors, and range from 3 to 100.

More specifically, we have a set of K = {3, 5, 7, 13, 21, 51, 100}, and we have executed i times the 10-Fold Cross Validation for all the $k_i \in K$.

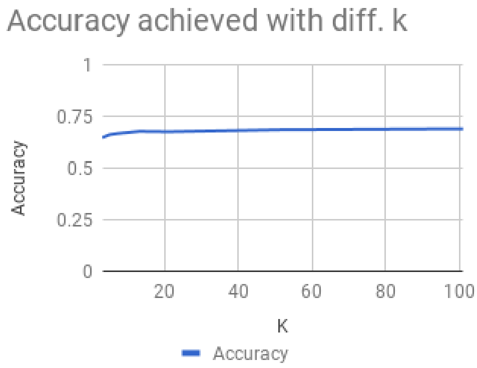
We have chosen to apply a Cross Validation with 10 folds because it seems reasonable for us to split the data of around 4.000 rows into 10 subsets and then cross validate each of them with the other ones.

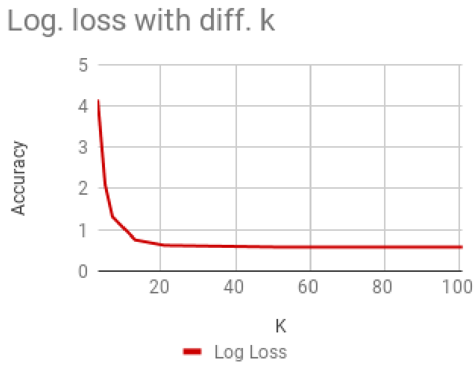
5.3 KNN executions

With our dataset, and for these different Ks, our executions give this accuracy and Log loss:

10-Fold Cross Validation on KNN

K	Acc.	Acc. Std	Log Loss
3	0.6459	0.0273	4.1508
5	0.6606	0.0354	2.1053
7	0.6657	0.0344	1.3162
13	0.6766	0.0321	0.7586
21	**0.6749**	**0.0317**	**0.6246**
51	0.6845	0.0326	0.5847
101	0.6891	0.0363	0.5842





From the table and graphics above, we can see that the log loss starts to stabilize when $k \geq 21$. Since there is not much of a difference between $k = 51$ and $k = 101$, we think it is preferable to settle on either $k = 51$ or $k = 21$, for the computational cost increases the greater the k is. And most importantly, **locality is lost.**

So, in our eyes, it is preferable to settle on $k = 21$ and preserve locality with very little trade-off in accuracy in our dataset rather than have a slight more accuracy with a very big k when $k = 51$, implying that we start to lose locality.

Having settled on $k = 21$, the accuracy of our model would be somewhere around a 67.5% with very little standard deviation, and a small Log Loss value, which means that the noise has been reduced.

5.4 Support Vector Machines

The second method that we have applied is Support Vector machine, also known as SVM.

SVM is a supervised learning method that constructs a hyperplane, which are linear boundary in our case, that separates quite well the classes in a finite dimensional space for classification. Its main objective is to find the "maximum-margin hyperplane" that divides the group of points.

The criterion for building a two-class classifier, as it is in our case, is to maximize the width of the margin between the classes, where the margin is considered the empty area around the decision boundary, defined by the distance to the support vectors. In general, it is considered that the larger the margin, the lower the generalization error of the classifier. In some cases it is not very easy to find a linear hyperplane that separates the classes perfectly, so the C parameter is introduced to determine the influence of the misclassification on the objective function.

We have used the implementation of *SVC* from the *Scitkit Learn* package in Python. This has allowed us to focus on the execution of the linear SVC itself rather on implementing it.

5.5 Methodology on SVM

The methodology applied on SVM, as we only apply Linear SVM, also consists on **performing a 10 fold cross validation on our whole data, but in this case, for different values of** C.

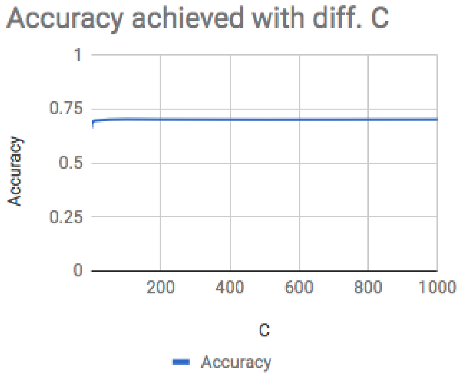
The selected values of C pretend to demonstrate the different expected behavior explained and find the value that best fit for our model. The range of C values chosen is C = { 0.1, 0.5, 1, 10, 50, 100, 200, 500, 1000 }, and we have executed i times the 10-Fold Cross Validation for all the $c_i \in C$.

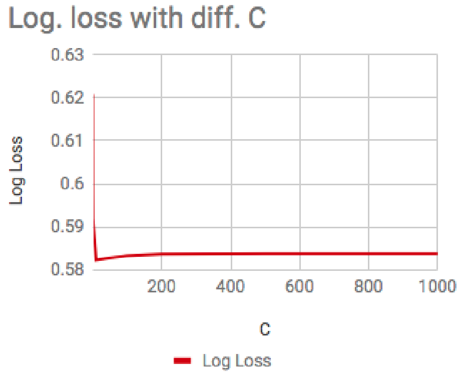
5.6 SVM executions

The results obtained after performing all the SVM executions of our preprocessed dataset with the different values of C defined are showed below.

10-Fold Cross Validation on SVM

C	Acc.	Acc. Std	Log Loss
0.1	0.6616	0.0409	0.6208
0.5	0.6823	0.0388	0.5991
1	0.6886	0.0334	0.592
10	**0.6954**	**0.0311**	**0.5823**
50	0.7007	0.0303	0.5828
100	0.7021	0.0329	0.5833
200	0.7012	0.0326	0.5837
500	0.7004	0.0326	0.5838
1000	0.7012	0.0321	0.5838

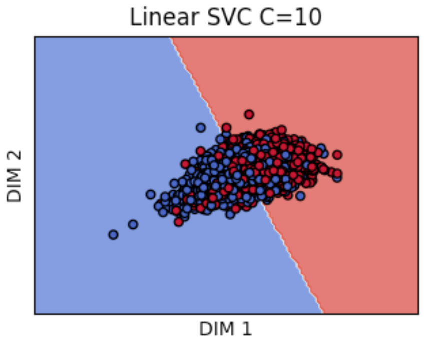


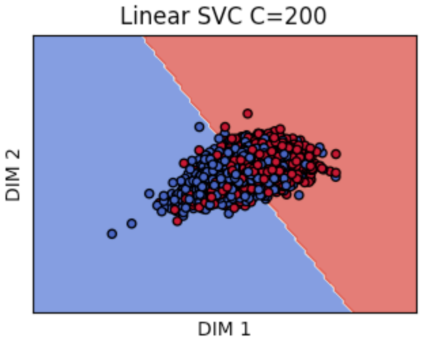


As we can see with the results retrieved on the table and graphics above, setting different values of C impact on the accuracy and log loss obtained for each execution. In fact, log loss stabilizes when $C > 10$ and no much improvement is obtained for larger values of C. That is why the optimal value of C parameter for our model is $C = 10$, as it minimizes the logarithmic loss and it obtains quite good accuracy (aprox. 0.69).

So, we can realize that the larger the value of C, the lower the margin is and, as a consequence, less missclassification is obtained because there is more flexibility to find a suitable hyperplane for separating the classes, but, up to a point that it may lead to overfitting the model. On the other hand, the smaller the value of C, the larger the margin is so more missclassification is obtained.

The different hyperplanes calculated when $C = 1$ and $C = 500$ are showed below. Note that it is just a representation on the first two dimensions to be able to see an intuitive result of the different hyperplanes obtained as we don't have information about the inertia that this plane really represents. But we can see that both models retrieve the same points for the same training data and that there is no much sparcity between both classes.





5.7 Convolutional Neural Networks

And finally, the last model used is Convolutional Neural Networks, also known as CNN.

Although having tried the MLP method provided in the *sci-kit* learn module, we decided to use a more customizable environment to work with CNN were we could have a more real hands-on with the parameters and neurons. So, *Keras* was used as a wrapper on top of the *Tensorflow* framework in order to simplify the modeling of the neural network.

5.7.1 Methodology on CNN

In order to ensure the optimality of the model in regards to the training methodology, several things are needed to be taken into account:

- The individuals used are exactly the same as the ones used on the other two methods.

- All the images are cropped, maintaining only their relevant region of the face and a posterior feature extraction based on LBP is also applied.

Then, this resulting set has been divided into 3 subsets:

- A first subset for training data, representing the 80% of the set.
- A second subset for testing data, representing the 20% of the set.
- A third subset for validation, representing the 10% over the previous training and testing set.

This third subset takes an important role on both, the training and the model selection. On each epoch of the training, the model is used to predict unseen data (validation). This allows us to have two loss metrics (calculated as a *log_ loss*), training loss and validation loss. Validation loss will be the indicator for model optimality. In order to make sure that no further training is done after the validation_loss has reached its minimum, an early stopping method and a checkpoint are applied in order to make sure that the resulting model is optimal.

Once the sets are generated, we need to specify several parameters that will define the characteristics of the convolutional neural network.

5.7.2 Parameter fine-tunning

To be able to create a neural network, several parameters are needed to be specified:

Convolutional layers

Although there is no general method for adjusting the parameters (kernel size or stride) for each of the convolutional layers[3], some considerations must be taken in order to determine those values.

A first "rule of thumb" that must be considered is to take a look at previous known and successfully implementations of an already used architecture (AlexNet, LeNet, MNIST) that suits our stated problem of gender classification.

Since the first **convolutional layer** tends to learn the features of an image in a high level perspective, a larger kernel size could/should be used. The usage of 3x3, 5x5 or 7x7 filters is quite common.

However, on a deeper (or lower) convolutional layers, the dimension of the feature maps are smaller (due to the downscaling of the pooling layers). Therefore, smaller kernel size for the convolutions must be applied since two pixels of a feature map are less correlated than ones from upper layers.

One clear advantage of a multiple layer network is the ability of feature learn on each step. In an image classification problem based on a CNN usually the first layer will be able to recognize basic features of the input set as its edges, the second layer could be able to differentiate between different types of edges (or shape) and the next layers could end up classifying between different collections of the previous shapes. Having a network with multiple layers could be beneficial, since they could learn all the necessary features of the input data and resulting as a good and generalized model.

Each layer is **initialized with a random weights** that are based in a uniform distribution limited bounded with a certain limit (glorot uniform). Weights are initialized (instead of being 0's), in order to make sure that the activation function provides an early activation for that neuron. By reference, the used **activation function** is a ReLu[7], which acts as a rectifier (converts negative vales into 0 and maintains positive ones).

After each convolutional layer a **pooling layer** is added in order to downsize (by 2 x 2) the spatial size of the feature maps extracted by the convolutions, resulting in a reduced computational cost of the network and overfitting control. The most common used function applied on the pooling is the maximum, that takes the input dimension and downsizes while applying the maximum on each downsized area.[5] The average can also be used in certain scenarios which don't benefit as much from different edges and shapes.

Finally, a **dropout** is added, where several units of each layer are removed (or not) depending on an input probability factor. This helps, for each epoch of training to avoid having all the units learn too much and therefore, specialize with the given input data at training time, in other words, it avoids the overfitting.

After defining the structure, we have trained our network on different number of layers, in order to see which depth and kernel translate in the minimum log loss. The different number of hidden layers and the kernel sizes of the convolutional layers tried are showed below.

Layers (kernel size)	**val. loss**	**val. accuracy**
(5,5)	0.3	86.45 %
(5,5) + (3,3)	0.263	89.26%
(5,5) + (3,3) + (3,3)	0.288	88.55%
(5,5) + (5,5) + (3,3) + (3,3)	0.2826	87.65%

The validation log loss of the previous table show us that the best number of layers that the model benefits from is two. Having seen that other layer configurations do not penalize too much the validation loss, we can conclude that, for the problem that we are trying to solve (and according to the input set) depth does not penalize the model as much as expected.

Layer wideness

Given our initial results based on our initial neural network, we want to explore the idea of enhancing our model to increase fitness by increasing layers width or the total network depth.

If we consider first the idea of increasing the layers width, each convolutional layer would result on more feature maps than before and therefore, more connections between one layer and another. By doing so, a wider model could end up memorizing all the features that we want to distinguish from the features of the input data. However, if the input set is not big enough to provide all the necessary diversity for the model to memorize, the training of the model will result in an overfitted instead of generalized one.

Unfortunately, this approach will never be entirely correct unless we can provide to the model all the possible features based on images at training time but then the training time will increase.

So, we have tried to find the most suitable value for the number of units in the layers, taking into account the previous layer configuration:

# units p. layer	**val. loss**	**val. accuracy**
8	0.3	86.45 %
16	0.28	87.5%
32	0.276	87.6%
64	0.254	91.27%
128	0.303	87.35%

Looking into a feasible number of units per layer (ranging from 8 to 128) we have found that medium size layers give us the best results (in terms of minimizing the validation log loss). But we need to take into account that increasing the number of layers could penalize both for overfitting and the training time.

Optimizer

Being the minimization the loss function the first objective for a neural network, several algorithms can be used in order to do so. Those are called the optimizers.

Keras provides different optimizers: SGD, RMSProp, AdaGrad, AdaDelta and Adam. Although most of them have their own hyper parameters that can be fine-tunned, no insight is going to be given because of pages limitation.

Although their differences, here is no one clear conclusion of which algorithm to choose and when. Each of them performs differently depending on the problem and parametrization. So, we aim to find the optimizer that works best for us:

Optimizer	**val. loss**	**val. accuracy**
SGD	0.3	89.46%
RMSProp	8.16	52.71%
AdaGrad	7.98	50.4%
AdaDelta	0.261	87.5%
Adam	**0.219**	**93.3%**

The results retrieved show Adam as the best performing. This does not surprise us, since it computes adaptive learning rates for each parameter, instead of specifying them statically.

Batch size

Another important parameter that needs to be analyzed is the batch size, that defines number of samples that are going to be through the network in one forward / backward pass. Usually this implies that a higher batch size will provide more examples for a single pass but also increase the memory usage. Note that usually a smaller batch size will result in a better generalization. [4]

Several runs changing the batch size (averaged in 3 executions each time) have given us the following results:

Batch size	**val.loss**	**val. accuracy**
1	0.37	86.75 %
32	**0.3**	**87.1%**
128	0.35	86.3%
256	0.7	52.7%

Having tried different values for the batch size, we have found that for batch size being the smallest (bs = 1) learning is impossible, which lead to loss values of $\tilde{8}$ and an accuracy of 50%. On the other side, having a really large batch size, such as 256, penalizes the ability to generalize keeping a constant validation loss and making the model stop training. Finally, for average values of batch size (32 - 128) the best results, in terms of minimizing validation log loss, are obtained.

5.8 Final model

Once having found all the best parameters for our problem, both the architecture and the metrics of the final model are shown. As the number of layers defined for our network is not high, we can see it as a simple model in terms of complexity. For this reason, the required number of epochs in order to reach a plateau in the validation loss remains low. This have favored us in both: time spent training the model ($\tilde{3}$0s.) as well as allowing us to try different parameterizations. Therefore, those multiple parameterizations have granted us a quite low log loss as well as a considerable accuracy.

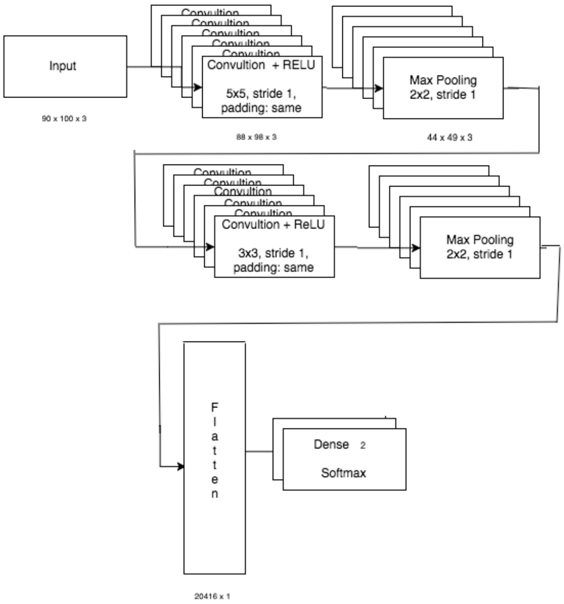


Figure 8: Final model

The following plot shows the performance of an execution of our model with the best parametrization found during the previous section. Keeping aside the train loss, we find the optimal model in the 9th epoch, which corresponds to the one which lower log loss.

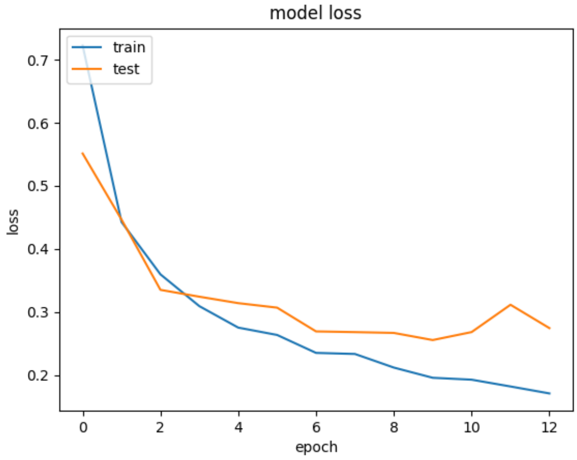


Figure 9: Final model loss

By having a checkpoint after every epoch, we make sure to stop training as well as saving the model if the log loss does not decrease for the next 3 epoch. This improvements have allowed us to find the best model given all the previous discussed features of a CNN that Keras and Tensorflow offer.

6 Comparison of results

Having applied all three models, now we want to compare their performance:

Method	Accuracy	Log loss
KNN	67.49%	0.62
SVM	69.56%	0.58
CNN	88%	0.258

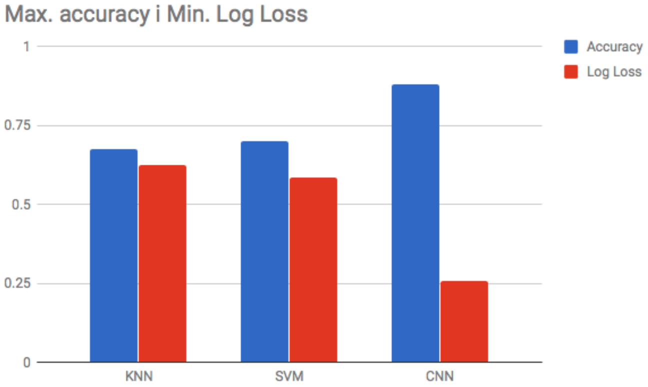


Figure 10: Final Results

As stated at the beginning of the section *Classification methods applied*, the best implemented model with which we obtain a better image gender classification is the one that minimizes the logarithmic loss and maximizes the accuracy. So, we can see with the results and grahics displayed, that in both terms, minimum log. loss and maximum accuracy, the developed **CNN model** is the best model for classifying our data. It improves much its performance from the other two models (almost 20 % more of accuracy) and there is not much difference between the other two developed models (not more than 2% of accuracy).

To sum up, the order from better to worst model is the following one.

1. CNN
2. SVM
3. KNN

7 Conclusions and future work

The conclusions gathered after having done this project and the future work that can be done are explained below.

7.1 Conclusions

Our main conclusions are the following ones:

- First of all, we both **underestimated and misunderstood, on our project proposal, the difficulty on features extraction**, as well as, how it was done. We expected a tool to be able to gather information from people faces such like how long their hair was, how big their eyes are or if a nose is crooked or not. In reality, feature extraction is very different, as it has been explained on this report.
- We also realized that image representation can be converted to numerical representation, which is the ones that is finally used for the classification models. It seems to be a cutting-edge topic, as models investigated are quite new and they have evolved from each other.
- Preprocessing has a huge impact on the precision of the results obtained for all the methods. Even it has not been reported, we realized that adding the cropping to the images for just comparing the relevant region of each image, significantly increase the results for all developed methods.
- Local Binary Patterns as is, with a uniform pattern applied has given us results good enough for us, with the accuracy being at least 65% or more. Our images had very different colors and lighting effects, but it did not seem to affect it much. Because of this, LBP has proved to be a reliable method on facial feature extraction.
- Choosing adequate methods for the data that wants to be classified or predicted and, more concretely, setting accurate parameter on each of them is the key to obtain the best possible results.
- *Scitkit Learn* package in Python has a huge variety of methods and facilities for developing machine learning experiments, as it offers a ready-to-use algorithms with multiple configurable parameters for each of them.
- Even it has not been explicitly reported, during the implementation of SVM, we realized that using kernalization for finding linear hyperplanes on other spaces does not always improve neither the log.loss nor the accuracy of the results retrieved with linear SVC. For example, linear SVM retrieved better results than rbf kernel. The most important thing is to successfully set the value of parameter C.
- We have confirmed that we do not have just to evaluate a model for how well it fits on our data (maximize the accuracy), we also have to calculate and reduce as much as possible its empirical error (minimize logarithmic loss).
- The implementation of crossvalidation on our developed KNN and SVM helped us to create a more generalized and non-overfitted model, as it can be seen on the results obtained.
- Although CNN tend to perform better (if a good preprocessing is done), are quite dependent on the parametrization that is applied.
- As stated, CNN is the most efficient method for our gender image classification. Taking a look on the missclassified images, first we see that there are some pictures that contain other objects that overlap the faces. We also came out that several images of males not well classified, and predicted as female, the male that appears on it has his mouth wide open, as the example showed below this lines. We could believe that it happens because females that appear on the set of images of the dataset tend to smile more or have bigger mouths than males.

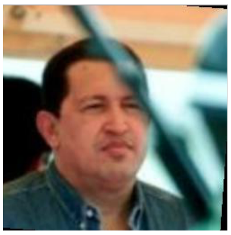


Figure 11:
Object overlapping

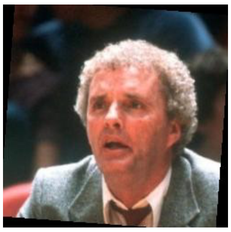


Figure 12:
Mouth wide open

7.2 Future work

For future work, the following things could be done.

- Improve LBP. We applied LBP "as is", with a uniform pattern, but LBP could be improved by trying different patterns and see how their accuracy changes. However, this improvement was out of scope of our project and we decided against it. After all, our project is about gender image classification, not feature extraction on facial images.
- Study how color impacts the performance of our classifiers. As we reasoned at the start of this report, color should not determine the gender of a person. Nevertheless, it would be very interesting to see how it affects on the performance of the three classifiers.
- Try other methods other than LBP. Although LBP has proved to be reliable, it would be interesting to see methods on images feature extraction that work in a very different way than LBP does.
- Perform other types of features classifications on the same dataset, such as smiling, ethnics, etc.
- Improve the crop of the images, by finding the pixels of the face outline on each picture and resizing the obtained faces.

  So, as we can see, most of them are related with the preprocessing or features extraction of the images.

References

[1] Balakrishna Gudla, Srinivasarao Chalamala, and Santosh Jami. “Local Binary Patterns for Gender Classification”. In: (Dec. 2015), pp. 19–22.

[2] di Huang et al. “Local Binary Patterns and Its Application to Facial Image Analysis: A Survey”. In: 41 (Nov. 2011), pp. 765–781.

[3] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition.* URL: `http://cs231n.github.io/convolutional-networks/`.

[4] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: `1609.04836`. URL: `http://arxiv.org/abs/1609.04836`.

[5] Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. “Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree”. In: 51 (2015). ISSN: 0162-8828. DOI: `10.1109/TPAMI.2017.2703082`. arXiv: `1509.08985`. URL: `http://arxiv.org/abs/1509.08985`.

[6] University of Massachussets. *Labeled Faces in the Wild Home.* Nov. 2007. URL: `http://vis-www.cs.umass.edu/lfw/`.

[7] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning* 3 (2010), pp. 807–814. ISSN: 1935-8237. DOI: `10.1.1.165.6419`. arXiv: `1111.6189v1`.

[8] Timo Ojala, Matti Pietikäinen, and T Maenpaa. “Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns”. In: 24 (Aug. 2002), pp. 971–987.

[9] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: (2017), pp. 1–13. arXiv: `1710.05941`. URL: `http://arxiv.org/abs/1710.05941`.

[10] Abbas Roayaei Ardakany, Mircea Nicolescu, and Monica Nicolescu. “An Extended Local Binary Pattern for Gender Classification”. In: (Dec. 2013), pp. 315–320.

[11] *scikit-learn.* URL: `http://scikit-learn.org/stable/index.html`.

[12] Caifeng Shan. “Learning local binary patterns for gender classification on real-world face images”. In: *Pattern Recognition Letters* 33.4 (2012). Intelligent Multimedia Interactivity, pp. 431–437. ISSN: 0167-8655. DOI: `https://doi.org/10.1016/j.patrec.2011.05.016`. URL: `http://www.sciencedirect.com/science/article/pii/S0167865511001607`.

[13] Paulo Villegas. *No Title.* 2016.

[14] Philipp Wagner. “Face Recognition with Python”. In: (July 2012).