# Macintosh HD:Users:konstantinamalliari:Downloads:gender.jpg

# 

# ***Gender Categorization***

Members:

Aleksandra Gkiali: BAPT 1611

Athanasios Gkinakos: BAPT 1616

Paraskevi Koutsioumpa: BAPT 1618

Konstantina Malliari: BAPT 1625

Maria Salemi: BAPT 1610

Contents

[Gender Categorization 1](#_Toc492290301)

[1. Introduction 2](#_Toc492290302)

[2. Exploratory Analysis 2](#_Toc492290303)

[2.1 Preprocessing 2](#_Toc492290304)

[2.2 Loading and Visualizing the data 4](#_Toc492290305)

[3. Methodologies 4](#_Toc492290306)

[3.1 Logistic regression 4](#_Toc492290307)

[3.2 KNN 8](#_Toc492290308)

[3.3 Multi-Layer Perception (MLP) 9](#_Toc492290309)

[3.4 Regularization 12](#_Toc492290310)

[3.5 CNN Methodology 13](#_Toc492290311)

[3.5.1 Preliminaries 13](#_Toc492290312)

[3.5.2 Getting the dataset ready 13](#_Toc492290313)

[3.5.3 Parameter definition 15](#_Toc492290314)

[3.5.4 Model Creation 15](#_Toc492290315)

[3.5.5 Problems faced 17](#_Toc492290316)

[3.5.6 Results 18](#_Toc492290317)

[4. Team Members 20](#_Toc492290318)

[5. References 20](#_Toc492290319)

# 1. Introduction

The human face is one of the most important biometric traits. When we look at a person’s face, we observe various characteristics, such as gender, age, ethnicity etc. as well as the current state of mind through expressions. The purpose of our project is to provide algorithms that perform gender classification, that is, to tell the gender of a person according to his/her face. It is an easy task for humans, but a challenging one for computers. Gender classification could be of important value in human-computer interaction, such as personal identification. Also it is a useful preprocessing step for face recognition. The face image is used for classifying the gender, so the gender classification process can make face recognition twice as fast by reducing search time for recognizing the person. Other gender classification applications can be found in communication, security, law enforcement, demographics studies, psychiatry and education.

Gender categorization is a problem that has been well studied in the last few years in the AI and machine learning literature.

The focus of our project involves a dataset on simple binary gender labels for a large number of human faces. This dataset can be found in <https://www.crowdflower.com/data-for-everyone/> For this dataset we tested several approaches such as logistic regression, KNN, MLP and the more recent approach of deep learning. Out of these, the best results were obtained using the CNN deep learning methodology as we report in the following sections.

The structure of our report is as follows: we first describe the preprocessing steps that we had to take and provide some illustrative pictures relatively to our dataset. We then discuss separately the performance of each methodology that we tried and report our findings.

# 2. Exploratory Analysis

## 2.1 Preprocessing

For the preprocessing procedure we had to deal with a .csv file that contains the information about the images. This file had 63.984 rows. Within that file one could find various pieces of information for each image such as trusted judgments, last judgements\_seen, the gender of corresponding figure, the URL of the server where the images are stored etc. Out of all this, we removed all the redundant information and we kept only the Gender and URL columns.

Before downloading the images we processed the .csv file so as to bring it in a more convenient form for the sequel (for example we eliminated empty characters and delete double commas, see the beginning of the code in the file Data\_Download.py). After that, we created two different arrays. The first one held all the images and the other one the gender classification of each image. During this step we threw away the images where the label of the gender was “unsure” (see Figure 2) and downloaded the images and stored them as arrays in greyscale format. In this step we also had to face another issue that concerns the URLs that did not respond to our request. To overcome the problem we converted the textual message returned for the request of each image to a string. This way, we were able to observe which requests resulted in successful downloading.

An example of a greyscale image that resulted after preprocessing can be seen in Figure 1.



Figure : A 300x300 greyscale image within our dataset

This image is stored as an array and its representation is as follows:

array([[86, 86, 86, ..., 79, 79, 79],

[84, 84, 85, ..., 79, 79, 79],

[81, 82, 84, ..., 79, 78, 78],

...,

[13, 12, 11, ..., 14, 14, 14],

[13, 12, 11, ..., 13, 13, 13],

[13, 12, 11, ..., 13, 13, 13]], dtype=uint8).

Although all these preprocessing steps look elementary this process took a sufficient amount of time (approximately 20 hours). One of the reasons was that we had to download the images from the server as well as the fact that the number of images was quite high given the resources that we had. The final dataset that we produced was 4GB(54413 rows).

# 



Figure : An example of an image that corresponds to "unsure" gender

## 2.2 Loading and Visualizing the data

After loading the data we performed some simple checks to test that the loading procedure worked correctly. For example we checked the size of the list that contains all images and the size of the list that contains the labels (which had to be equal).

The next step we took in order to explore the data was to visualize the images. By doing this we observed that all images were not of the same size (see Figure 3). In particular the vast majority of the dataset contained images of size 300x300 and this was the maximum size. We therefor, dropped all images with smaller size.

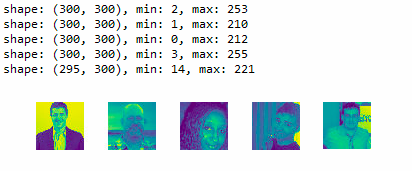


Figure : Examples from our dataset. Note that the last picture has a smaller size(295x300)

A final observation we would like to make is that the dataset was not very well balanced and contained a significantly higher number of men than women. Namely, out of the approximately 54,000 images, there were only 7302 female images. For this reason, we often had to select a smaller but more balanced subset of the data in our trials.

# 3. Methodologies

## 3.1 Logistic regression

In order to implement logistic regression with tensorflow, the images were firstly flattened, i.e., they were transformed from 300x300 to 1x90000. In addition, the classes were transformed from strings (“male”/“female”) to one-hot vectors. If the gender of an image was “male”, then its class was transformed into [1, 0]. If the gender was “female”, then it was transformed into [0, 1].

The next step was to split the images into a training set and a test set. Out of the 54,392 images, the first 50,000 were selected as the training set and the rest 4392 were selected as the test set. This means that the model was trained using the first 50000 images and, then, its performance was tested on the remaining 4392 images.

The general idea for the code used for the logistic regression was retrieved from the classic code found online for the MNIST dataset classification, with appropriate adjustments regarding the parameters, so that they correspond to the current project.

Two placeholders were defined, which were the input data (i.e., the images) and the classes (i.e. “male” or “female”). The first placeholder was a 2-D array with its first dimension undefined and its second dimension being of size 90000, calculated as the multiplication of the images’ height and width. The second placeholder was also a 2-D array with its first dimension unknown and its second dimension being of size 2, which corresponds to the number of different possible classes.

Two initial variables, “W” and “b” were defined, which were the beginning of the network graph. “W” was the 2-D Μ90000x2 matrix that held the weights of the network. In total, 180000 weights were calculated during each epoch – 2 for each element of each image’s array:

W = tf.Variable(tf.zeros([90000, 2]))

“b” was a 1-D array of size 2, which held the biases:

b = tf.Variable(tf.zeros([2]))

Both of these variables were initialized to zero, and it was tested whether a random initialization would yield better results, but that did not prove to be the case. Four additional variables were defined, which were all related to the two initial ones.

One of them was the prediction variable “pred”, which was defined as the softmax function of the matrix multiplication of the weights “W” and the inputs “x” plus the biases “b”:

pred = tf.nn.softmax(tf.matmul(x, W) + b)

The next variable defined was the cost. The cost function that was used is the cross entropy with logits, which is a very common measure of inconsistency and a very popular cost function for neural networks:

cost = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(pred), reduction\_indices=1))

The next choice to make, was the optimizing algorithm that was used for the model’s training epoch by epoch. Among the available optimizers in the literature, the one used by our team is the gradient descent optimizer. The task of the gradient descent is to simply find the “lowest” direction of the cost function at each step, and move towards that direction. This is not always perfect, since it is very common and usual that the algorithm will find a local minimum of the cost function, instead of the global minimum, but, still, in most of the cases its results are quite promising:

optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(cost)

The next variable was the one that defined which predictions were correct. Since the classes were represented by a vector of two elements ([1, 0] for male and [0, 1] for female), in order to check whether a prediction was correct, the system simply needed to identify whether the maximum value of the vector of the prediction was the same as the maximum value of the vector of the true class. If this held true, then, subsequently, both classes were either male or female, which would mean that the prediction was correct:

correct\_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))

The last variable defined for the network was its accuracy, which was simply set to be the mean value of the correct predictions. When a prediction was correct, then the “correct\_prediction” variable would return “1”, whereas, if it was wrong, it would return “0”. By calculating the mean of all these “correct\_predictions”, the system indirectly calculated the percentage of the correct results, i.e. the network’s accuracy:

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))

A parameter that played a very important role in the algorithm was the learning rate. A lot of trials were made with the learning rate, which is the parameter that indicates the size of the steps that the model will make towards the direction that the gradient of the cost indicates. It was observed that, when the learning rate was too big, then the cost resulted into null values. When the learning rate was too small, then the model would not learn as desired. Finally, after trial and error, it was found that a learning rate of 5\*10-9 yielded satisfactory results, i.e., the cost during the first epochs was high and, epoch by epoch, it deteriorated resulting in increasing training and test accuracies.

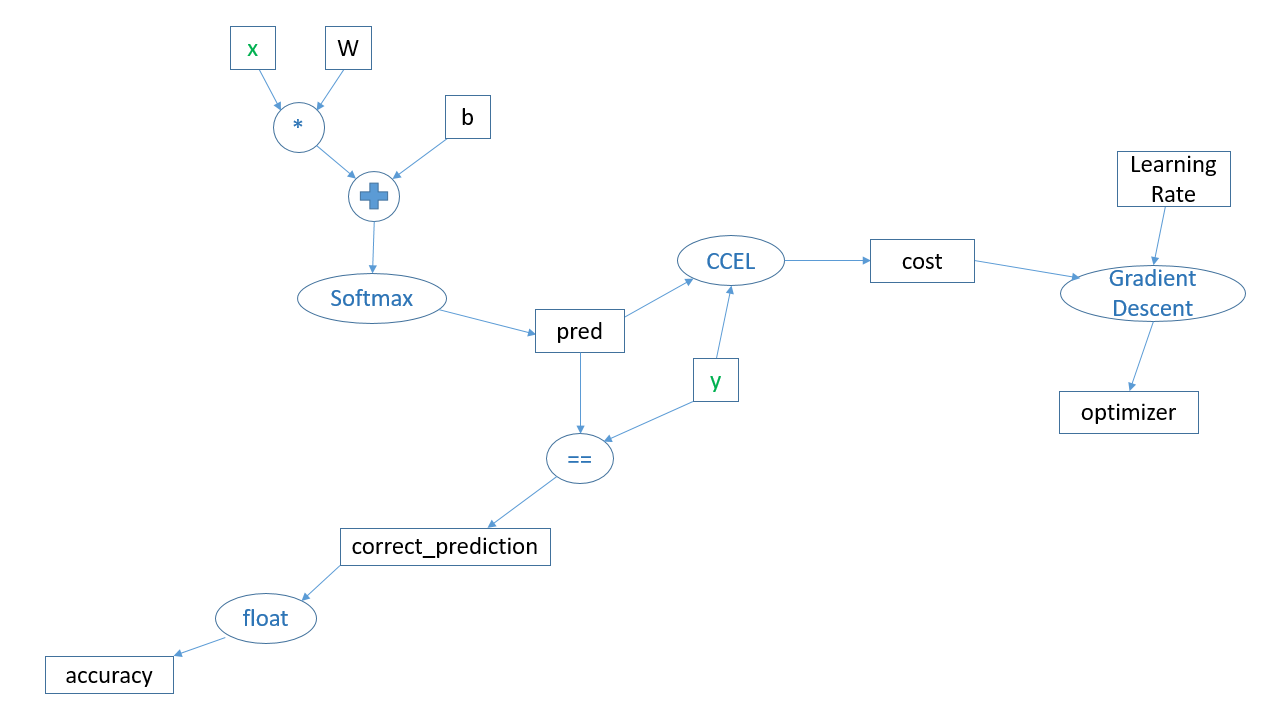


Figure : The neural network’s architecture graph. With green are the placeholders, with black the variables, and with blue the functions. CCEL is an abbreviation for Categorical Cross Entropy Loss.

After the definition of all the parameters, placeholders, and variables of the network’s graph, the next step was to actually run it. Within the session, the placeholders were fed with batches of 64 images during each epoch. These 64 images were chosen at random during each epoch, using the *random.shuffle* function of the *numpy* package. So, 64 images of the training set were fed into the graph during each epoch and, based on these, it trained step by step, i.e. it modified “W” and “b” so that the “cost” became less. The test accuracy of the network at each epoch was tested by feeding it with the test set of 4392 images. The maximum training accuracy that was accomplished was 87% and the maximum test accuracy was 83%. Although these percentages were not satisfactory, it was the best that could be accomplished with the current network.

Then, another trial was implemented, during which the training dataset was selected to be of equal number of images per sex. All the female images from the training ones were kept, which were 6500, and 6500 more males were also chosen. In total, out of the 50000 images for training, only the 13000 were used. Although this training dataset was much smaller, it was worth a trial to train the network equally in both sexes, so that it was not biased in favour of one of the two. The resulting train accuracy of this model was 64%, and the test accuracy 60%. These percentages were much lower than the ones accomplished previously, but they were significantly higher than pure luck (50%). In addition, if one takes into account the fact that about 80% of the dataset consisted of males, the 83% accuracy that was initially accomplished might not mean that much. A 60%, though, clearly shows that the network had learned from the data. It obviously had not learned much, but it had learned.

So, it was concluded that the network’s architecture was inadequate to describe the features of the dataset. This seemed quite expected, since the images of the dataset were more complex compared to the images of the MNIST dataset. Each photograph was taken from a different angle and different distance from the person, in contradiction to the numbers of the MNIST dataset, which were all at about the same size. In addition, a person’s face has a lot more features than an integer, even if this integer is handwritten. Thus, it was assumed that the training dataset was not big enough to train such a model.

Problems with learning rate

A lot of trials were implemented with different values for the learning rate of the logistic regression network. It was noticed that, if the learning rate was bigger than 10-8, the cost function would result in null values for every epoch. This indicated that, probably, the predictions for higher learning rates would take values close to zero and, thus, when put as a parameter into the logarithm during the cost function definition, it would yield null values, since the logarithm of zero is not defined:

cost = tf.reduce\_mean(-tf.reduce\_sum(y \* tf.log(pred), reduction\_indices=1))

When the learning rate became a lot smaller, i.e. smaller than 10-8, then the cost function would manage to take values epoch by epoch. After a lot of trial and error, it was decided that a learning rate of 5\*10-9 would yield the best possible results.

## 3.2 KNN

For the KNN model, it was impossible to use as much training data as with the logistic regression, due to limited hardware resources. Thus, the training was implemented with 5000 training images. These images were, again, flattened so that they were transformed from 300x300 to 1x90000 elements. The test accuracy of the KNN model was tested on 100 test images, different from the ones used to train it.

The implementation of the KNN network was simpler. Two placeholders were defined, one for the input images and one for their corresponding classes. Two were the variables as well, the distance and the prediction. The distance used was the Euclidean.

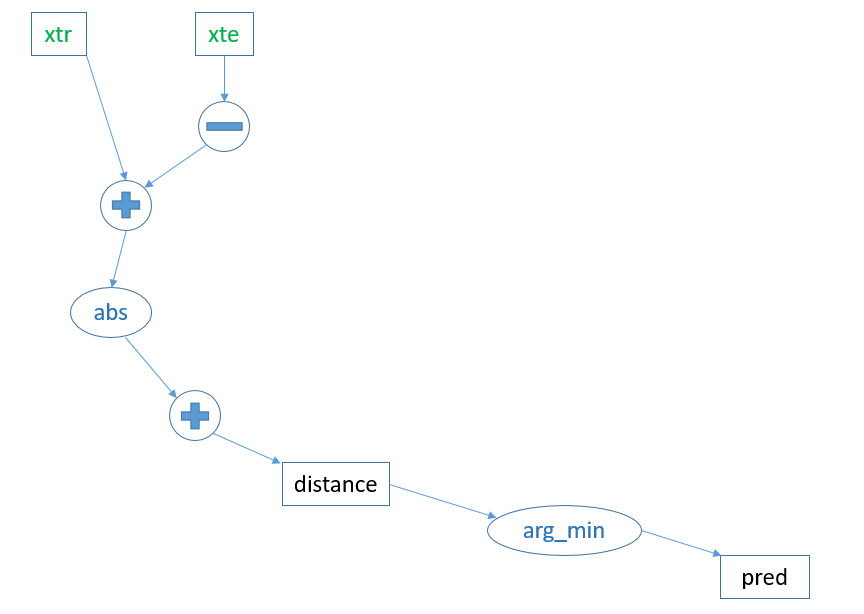


Figure : KNN network’s graph.

The accuracy accomplished was 81%, which was not even a little bit better than the one accomplished by the logistic regression model. It was thought that the reason for that was that the training data mostly consisted of images of class “male”, since the male images in the whole project’s dataset were much more than the females.

Subsequently, the same alternative approach as with the logistic regression was used, where the 5000 training images were not chosen at random, but they were forced to be of equal number in both classes, i.e., 2500 males and 2500 females. As a result of this approach, the test accuracy of the KNN model was 62%, which is certainly higher than pure luck, but, again, not that promising. This is a percentage that indicates that the network managed to model the images to some extent, but this extent was obviously very limited, although better than the corresponding one from logistic regression.

Training data size in ΚΝΝ

The main issue encountered during the implementation of the KNN neural network was the very high limitation regarding the size of the training dataset. This did not come as a great surprise to our team, since KNN demands the calculations of the distances between all the units at each step – an implementation that required a lot of computing power and resources to be accomplished within logical time. The characteristics of our computers allowed to have 5000 images at maximum as training dataset.

## 3.3 Multi-Layer Perception (MLP)

The MLP design allowed our network to go deep. During this implementation, hidden layers were added to the architecture of the neural network, so that more degrees of freedom could be supported, allowing the network to model much more features of the images.

The major change in the implementation of the network compared to the logistic regression (seen previously) was the definition of the “pred” variable. In logistic regression, this was simply defined as the softmax of the matrix multiplication of “x” with “W” plus the bias “b”. In the MLP, the prediction is defined through the multi-layer perceptron, which is basically the function that implemented the neural network with the input, hidden, and output layers respectively. The perceptron’s architecture can be seen in Figure 3, which depicts a deep neural network with 2 hidden layers. In the current implementation of our project, the number of hidden layers that we used was either 3 or 5.

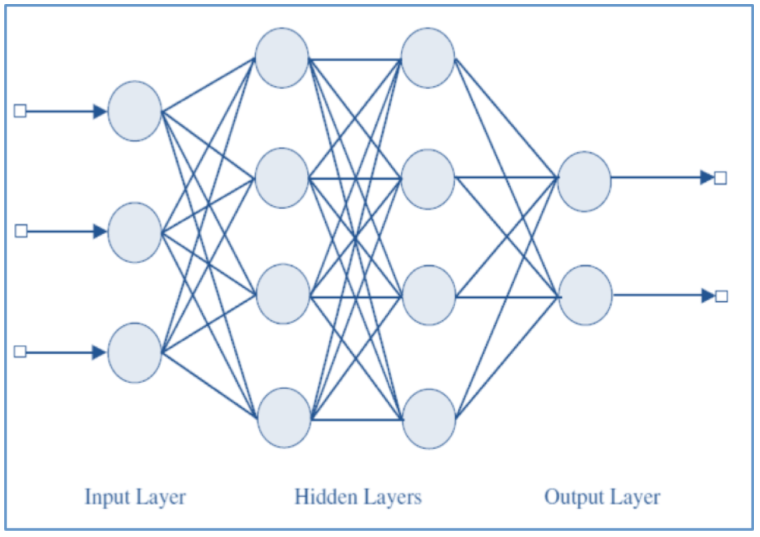


Figure : Multi-Layer Perceptron with 2 hidden layers. The first layer represents the input data “x”. The arrows represent the weights “W”. The circles represent the neurons. Each neuron is a function of its input. The neurons of the hidden layers represent the ReLU function, whereas the neurons of the output layer represent the softmax function.

For each of the hidden layers, the activation function used was ReLU (Rectified Linear Unit), and for the output layer, the activation function used was softmax.

Apart from the MLP, there were two additional slight differences from the logistic regression. Firstly, the cost function used was, again, the cross entropy, but this time the built-in in the tensorflow package function was chosen:

Cost = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred, labels=y))

Secondly, the optimizing algorithm used was the AdamOptimizer:

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate).minimize(cost)

Regarding the learning rate, it was noticed that a much bigger value compared to the logistic regression would yield satisfactory results.

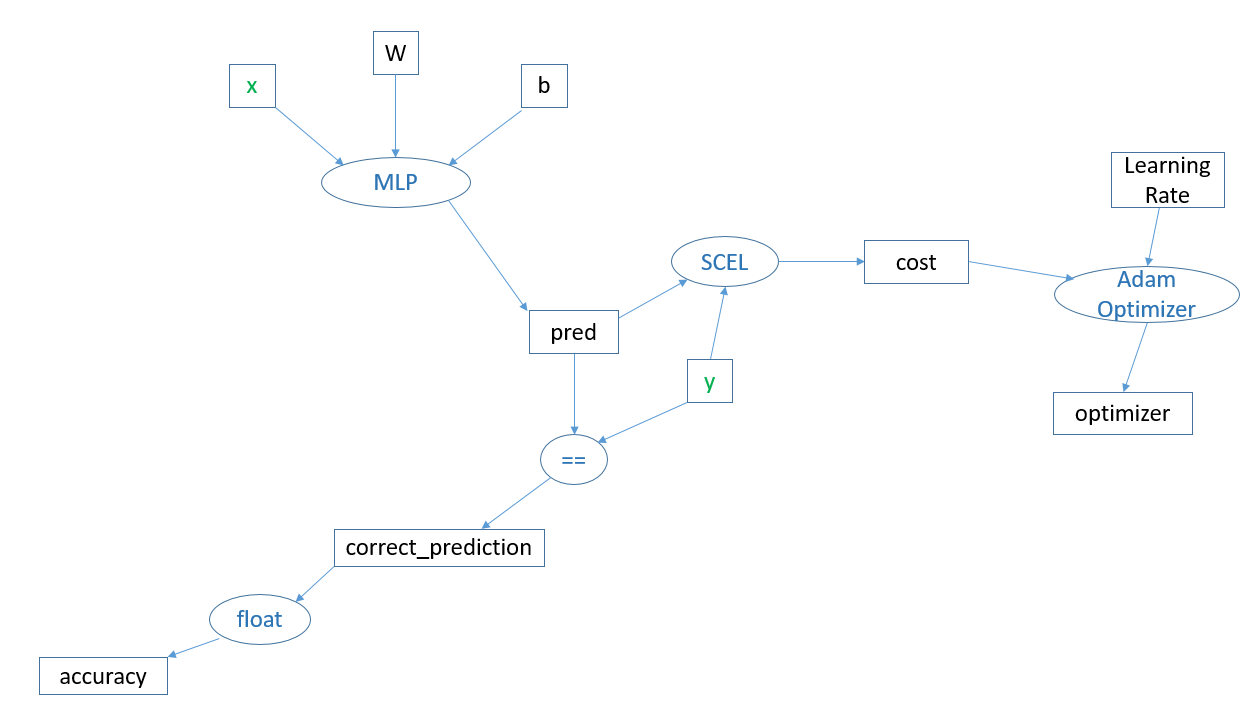


Figure : The deep neural network’s architecture graph. SCEL is an abbreviation for Softmax Cross Entropy with Logits.

Several trials were made with the parameters and the architecture of the network. Firstly, it was designed with 3 hidden layers, each one of them consisting of 60 neurons. The resulting accuracies of this network were 78% for the training data and 83% for the test data. These accuracies were not satisfactory, since they were even worse than the accuracies of the logistic regression and the KNN. Thus, we tried to change the number of neurons per layer to 256. So, the network now consisted of 3 hidden layers with 256 neurons each, allowing more degrees of freedom for the model. This model was computationally more complex for our computer to run and, thus, it took quite some time for each epoch. We let this model run for 40 epochs, and its training accuracy was better than the one with the 60 neurons (85%), but the test accuracy remained at 82-83%.

The next trial was to compress the images to arrays of size 100x100, so that the algorithm ran faster and it became possible to run it for many more iterations. After 60 epochs, the train accuracy accomplished was 87% and the test accuracy was 83% - percentages slightly better that the previous ones. The same accuracies were achieved when the same algorithm was run on 30x30 images, i.e., even more compressed. The speed of the process was much faster, though.

Furthermore, the network was slightly reconfigured so that it consisted of 5 hidden layers. This network only ran with compressed images (30x30), or else the available hardware would not be able to implement the process. Contrary to what one would expect, the resulting accuracies were not better than the previous architectures’. Actually, they were even worse – the training accuracy did not exceed 81% and the test accuracy fluctuated around 82-83% epoch by epoch, with the corresponding cost fluctuating around a local minimum as well.

Finally, in order to train the network with an equal number of male and female images, only 13100 images were chosen from the original training dataset – 6550 males and 6550 females. This way, it was believed that the network would have the same amount of information for both sexes and, thus, its results would not be that biased in favour of male. This, of course, came at the cost of having a much smaller training dataset for the neural network. Surprisingly enough, the train accuracy accomplished by this setting reached 100% after 3500 epochs. The test accuracy was significantly lower, though, not exceeding 58%. The 100% training accuracy accomplished by the network indicates that the cost function was minimized at the global minimum. In other words, the neural network managed to modify its parameters in such a way that it described the training images perfectly (100%). Despite this fact, the test accuracy was not exceptionally high. This probably means that the training dataset was not big enough, i.e. the 13100 images were not enough to describe the features of all the possible images that the model could have as input. Unfortunately, there were no additional images to train the model. So, in order to accomplish a significantly higher test accuracy, a completely different architecture needed to be investigated. Such an architecture is provided by the Convolutional Neural Networks (CNN), which are analysed in the next section.

Images Size in the MLP

The size of the images was the source of many technical limitations in general, but in the multi-layer models these limitations proved to be even more tangible. As more and more layers were added to the architecture of the deep neural network, and as the size of each layer became bigger, the 90000 array elements per image could not be processed by our computers. Transforming the arrays of the images to 100x100 or even 30x30 allowed the multi-layer networks to be designed with greater complexity and depth, thus providing them with much more degrees of freedom and making it possible for them to describe and model the numerous features of a person’s face.

## 3.4 Regularization

Regularization was a technical issue that our team encountered when constructing and implementing most of the networks. By regularization we are referring to the transformation of the values of each pixel from integers between 0 and 255 into floats between 0 and 1, which would indicate the percentage of grey existing in each pixel. This was practically accomplished by iterating through the whole dataset and dividing each element of the arrays with 255. But, when the networks were fed with these float values, our computers would seem completely inadequate to withstand the computation complexity, and either stop working and throw memory errors or even crash. Thus, it was decided for practical reasons that regularization would not be implemented.

## 3.5 CNN Methodology

### 3.5.1 Preliminaries

Another methodology used for gender recognition on the specific dataset is the Convolution Neural Networks or CNN. CNN is specialized in deep learning and is one of the most popular and efficient models for image categorization using neural networks.

CNN is based on multilayer architecture through which it learns information about the data given as input. Implementing filters on each layer enables the algorithm to scan and read the image and in combination with the use of special weights, CNN is able to interpret the characteristics of the face pictured, in order to compute whether a man or a woman is depicted. Furthermore, the cost is calculated, based on the “softmax” function, which is actually the difference between the real and the predicted classes. This measure indicates the accuracy and the effectiveness of the algorithm, in other words the precision of the output is based on the actual classes that each person belongs to.

For the dataset in question, CNN is applied in order to categorize the pictures into two classes, male or female. The model is trained on the total number of the pictures and then it is tested on the first images of the dataset (equal to the batch size) computing the precision of the prediction.

### 3.5.2 Getting the dataset ready

After the preprocessing of the initial dataset, the total number of images was 54.413. It is noteworthy that the pictures are already converted into tables in order for the program to process them suitably. Loading the input images in the main code, only those with dimensions 300x300 are kept, diminishing the total number into 54.392.

The same procedure is followed for the classes of the images as well. As already mentioned there are two classes; the male class corresponding to “0” and the female class corresponding to “1”. A significant change in the implementation was the conversion of the classes into “one-hot” vectors, in other words into a 2-Dl array filled with zeros and ones. The first column represents the men with ones only in the places where the images depict a man and the second column represents the women in the same way.

Continuing the process, the dimensions of the images are reduced to 100x100. This change is made due to the fact that the initial size of the pictures is very large and requires more RAM memory, which was not available by our resources. Thus, diminishing the size of the input enables the program to be executed faster and more efficiently. Furthermore, it also allows us to use a higher number of training and test data. Namely, the training dataset was increased to the total number of available images.

To accomplish the size reduction, we used the “resize” command of Python, and we applied it on each picture itself and not on the arrays in which it had been transformed. Applying the command directly on the arrays forces the program to focus on a specific part of the picture, disregarding a great percentage of information, as shown in Figures 8 and 9.



Figure : Initial dimensions 300x300

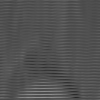


Figure : Array transformation 100x100

On the contrary, applying the command directly on the image yields more decent results as the only parameter affected is the image analysis (see Figure 10).

Figure 8: Initial dimensions 300x300 Figure : Image transformation 100x100

### 3.5.3 Parameter definition

Several parameters are defined in the program affecting directly the regular prosecution of the algorithm. To be more specific, the first parameter that needs to be set is the *learning rate*. Its purpose is to control the rate with which the algorithm will gradually converge to the best point of interest. Many values were tested on this parameter as it influences the results to a great extent.

The next point that has to be defined is the *training iterations*. Its value corresponds to the total number of iterations that the algorithm will execute. When CNN reaches this number the processing ends.

The *batch size* is the size of the input that the algorithm will be trained on during the iterations. In other words, the batch consists of randomly selected pictures from the training dataset. For the analysis, we used several values in order to record its impact and achieve better results.

After that the *display step* is set, which defines the frequency with which the results are printed.

Finally, the parameter *dropout* is fixed which helps in combatting overfitting by rejecting the nodes of the networks with percentage less than 75%.

For the creation of the model, it is also necessary to define the *filters* that will be applied on the images. Their dimensions are set in 5x5, width and depth respectively. One color channel is used as the images are processed in grey scale, and finally we tried different values for the number of filters used per layer. In the analysis, 64, 128, 256 and 512 filters are used in each layer. Their creation is necessary as they control the calculation of the weights on which the algorithm is based for the gender assessment.

### 3.5.4 Model Creation

After the preprocessing of the data that will be given as input in the code and the definition of the parameters for the efficient implementation of the algorithm, follows the training and the testing part of CNN.

First of all the “convolution layers” are created. These layers consist of filters, which process the image computing the dot product between the entries of the filter and the input images. Filters progressively “learn” using weights and when specific characteristics are detected they are activated. The sum of these activated filters forms the output volume of the convolution layer which is used as input in the next layer. A visual representation is given in Figure 11.

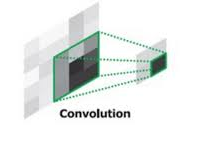


Figure : Convolution layers

Another important part of CNN constitutes the “pooling layers”. During pooling processing, the pictures are divided into smaller pieces using filters of size 2x2. The algorithm processes each one of them and keeps only the rectangle with the maximum value (max pooling function, Figure 12). The pooling layer serves to progressively reduce the number of parameters and amount of computation in the network, and also to control overfitting. Commonly they are used between two convolution layers.



Figure : Max pooling example

After the completion of all computations in each layer the “ReLU” function is activated. ReLU takes as input the output of the convolution layer and decides whether the figure depicted in the picture is a man or a woman.

The convolution and the pooling layers are used both in the training and testing process. In Figure 13, the flow of these layers is depicted.

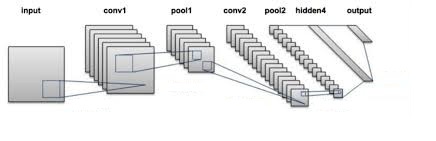


Figure : CNN processing

Furthermore, the model computes a cost, which measures the difference between the predicted and the real classes in the test set. The function used for this purpose is called “softmax”. Moreover, an optimizer for the improvement of the cost is defined based on the learning rate mentioned before.

At the beginning of the iterations, the batch size on which the algorithm will be trained is created containing pictures of both sexes. After that, CNN processes the images through the layers, learns the characteristics that correspond to the genders, keeps the weights calculated using the filters and computes the loss and the training cost. Then it reads the test images, repeats the steps applying the same filters on them, guesses the classes and calculates the testing accuracy. The program ends when the fixed iteration number is exceeded.

### 3.5.5 Problems faced

#### Data structure

One of the major problems encountered was the structure of the dataset as the total number of men is significantly larger than the number of women in the initial dataset. Specifically there are 47111 males and only 7302 females in the input dataset. This led to several problems as CNN learnt only characteristics of men and could not predict women. In order to solve this problem the batch is selected in purpose so that 25% of its size corresponds to women. The rest 75% is chosen randomly from the whole dataset. Thus, it is ensured that a decent number of both sexes are selected and CNN can process men and women equally well during the training part.

#### Complexity of input images

Another problem faced was the fact that the images are convoluted. In other words, they depict faces in different ways and the algorithm has to deal with facial expressions and body postures. A solution to this matter is to increase the number of convolution layers. As a result, each image is processed in deepest levels and CNN is able to understand better the characteristics of the sexes.

#### Resolution of images

The size of the pictures also put constrains in the execution. Processing images with dimensions 300x300 required lots of resources and especially RAM memory, which was not available by our existing resources. The runs lasted hours, the algorithm could not even be trained on the total number of the images and the testing set was too small to reach a decent output. For all these reasons, the dimensions of the images were converted into 100x100 as already mentioned, so as to avoid all these critical issues.

#### Selection of Learning Rate

Last but not least, the learning rate in combination with the structure of the dataset posed a problem in the code. Very small values of the learning rate led the algorithm to learn at a very slow rate, requiring a really big number of iterations in order to reach useful results. Certainly, this option was not preferred as it demanded a lot of time and resources. On the other hand, big values of the learning rate led to poor outcomes, as the algorithm could not converge to a point and eventually learn from the data. Therefore, a big range of values was tested in order to understand how it affects the results taking into consideration the rest of the parameters as well.

### 3.5.6 Results

In order to compare the results and draw interesting conclusions several runs were made changing each time one of the parameters and keeping the rest fixed.

#### Learning rate

To begin with, the *learning rate* is one of the parameters which display variances in the output. For the analysis, we have recorded four different values of learning rate, despite the fact that we made several more trials.

The parameters which remained stable in the runs were:

* 4 convolution layers
* 100000 training iterations
* 128 batch size
* 50 display step
* 2/3 men and 1/3 women for the training

Using learning rate = 0.1, we observed that the model did not meet our expectations. The training accuracy was approximately 57% in each trial and the loss showed a fluctuation instead of a gradual decrease. Furthermore, the testing accuracy reached 96% indicating that the algorithm could predict only the male pictures ignoring the presence of women in the dataset (960 out of 1000 are men in the testing dataset).

As the value of learning rate was diminishing, the results showed a decent improvement. When its value was set to 0.01, there was a slight increase in the training accuracy (70%), the loss was continually decreasing and the testing accuracy reached 83% of success.

Even smaller values of learning rate, equal to 0.001, yielded better results in the training and testing data, 74% and 84% respectively. However, it is noteworthy that the loss was decreasing with very slow rate requiring a greater number of iterations to converge.

It becomes clear that big values of learning rate do not help CNN to learn both sexes while very small values give better percentages, yet demand many training iterations in order to predict correctly the classes.

#### Convolution layers

The number of *convolution layers* also played an important role in the results. The parameters that remained fixed were:

* 0.001 learning rate
* 100000 training iterations
* 128 batch size
* 50 display step
* 2/3 men and 1/3 women for the training

At first, 2 convolution layers were used. The results were satisfying enough as the training accuracy reached 67%, while the testing accuracy reached 76% of success. Increasing the number of convolutions into 4 and 5, the output was even better. The training accuracy was approximately 75% and the testing accuracy 80%. A significant difference between the trials is that when the number of convolution layers was bigger, the loss decreased with very slow rate, leading to the conclusion that the number of training iterations needs to be raised. There is a high probability of achieving really impressive results if this increase is applied. However, due to the cost and the time that this trial required, we were not able to implement it.

To sum up, an increase in the number of convolution layers helps CNN to learn better. This is anticipated considering that processing an image in more detail reveals new information about the people depicted in the images and thus gives the opportunity of better predictions.

#### Batch size

Another parameter that was tested is the *batch size*. The batch size contributes to the training part of the algorithm selecting randomly the pictures on which CNN will be trained in each repetition.

The fixed parameters were the following:

* 0.001 learning rate
* 100000 training iterations
* 4 convolution layers
* 50 display step
* 2/3 men and 1/3 women for the training

The numbers used for the batch size were 64, 128 and 256 respectively. The differences in the results were minor, running the code for 100000 iterations. The training accuracy fluctuated around the same percentages in all three cases while the testing accuracy showed a very small increase when the batch size was getting bigger, specifically from 72% to 84% and 74% respectively. As far as the loss is concerned, bigger batch size once again requires a bigger number of training iterations as its decrease occurs in slower rate.

In other words, the batch size did not seem to affect to a large extent the results, keeping in mind that the rest of the parameters were kept fixed. Nevertheless, the output was slightly better when its size was equal to 128.

#### Percentages of men and women in the training dataset

The last parameter that was taken into consideration is the percentage of men and women that CNN was based on, for training. As already mentioned, the number of men in the dataset is significantly larger than the number of women, which set a problem in the training session of the algorithm. The bigger part of the batch consisted of male cases as the images were selected randomly and thus, there was no sufficient information about women. The issue was settled by forcing the code to select a specific part of the batch size only from the female pictures.

The parameters that remained stable were:

* 0.001 learning rate
* 100000 training iterations
* 128 batch size
* 4 convolution layers
* 50 display step

The initial split made was 50% men and 50% women. According to the training accuracy, it seemed that the algorithm gradually learnt reaching an accuracy of 74%. However, the testing accuracy stayed in low rates as there was approximately 51% of success.

The second choice was to split the batch into a 2/3 fraction of men and 1/3 fraction of women. This option led to better results, since the training accuracy was 70%, as well as before, but in the test set the accuracy reached 84% of success.

It becomes clear that even the number of cases for each sex plays an important role in the model. This shows the complexity and the perplexity of the algorithm. CNN is an efficient model as long as all the parameters are carefully determined and there is a large number of input data.

# 4. Team Members

The team for this report consisted of 5 members.

Alexandra Gkiali: holds a B.Sc. degree from the Department of Management Science and Technology of AUEB. She works as an E-business customer service officer in Eurobank.

Athanasios Gkinakos: holds a Diploma from the School of Electrical and Computer Engineering of NTUA. He is currently a consultant for the company i2s SA.

Paraskevi Koutsioumpa: holds a B.Sc degree from the Department of Informatics of AUEB. She is currently a mainframe developer in banking industry for IBM.

Konstantina Malliari: holds a B.Sc degree from the Department of Informatics of AUEB. She works as an ETL developer in the banking sector for IBM.

Maria Salemi: holds a B.Sc degree from the Department of Informatics of AUEB. She works as an ETL developer in the banking sector for IBM.

# 5. References

[1] <https://www.tensorflow.org/tutorials/deep_cnn>

[2] <https://en.wikipedia.org/wiki/TensorFlow>

[3] [https://www.datacamp.com/community/tutorials/tensorflow-tutorial - gs.syTHxLk](https://www.datacamp.com/community/tutorials/tensorflow-tutorial#gs.syTHxLk)

[4] TensorFlow Labs Material

[5] G. Levi and T. Hassner, *Age and Gender Classification using Convolutional Neural Networks*, IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG), at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 2015.