Machine Learning – project:

Gender Classification

classification using 4 Machine Learning techniques. Convolutional neural network (CNN), Multilayer Perceptron (MLP), Logistic Regression, k-nearest neighbors (KNN).

<https://github.com/TalNoamX/Gender-Classification>

**Introduction:**

In this paper, I will cover my work on this project, which dataset I used, techniques, libraries, code, progress, difficulties, and results.

I choose to make my project about classification which is more relevant to deep learning techniques, in more specific "Gender Classification", my models are built to classify between men and women.

I used 4 techniques: KNN, logistic regression, MLP, and CNN.

I tried to build my models as accurate as possible, to properly classify men and women using a dataset of over 100k images. All the models run on my laptop, which was easy at first with the simple models like logistic regression and the KNN, but pretty hard when it comes to MLP and CNN but more on that later.

**Dataset:**

This dataset I use was download from the Kaggle website. It contains more than 100k of images of female and male portraits, and already divided into 3 categories: train, test, and validation. For example:

The images for the logistic regression, and MLP models, I turned grayscale and size 50x50, the reason is that the models are not complex enough for the RGB color system. Example:

For the KNN model, I also used grayscale images but because the model was simpler I kept the size 200x200.

And for the CNN model, I used colored images of the size 200x200, the CNN model is enough complex for regular size images and color.

I resize all the images to the same size just to fit the model settings, to simplify the process, and to increase the pace of learning.

**k-nearest neighbors:**

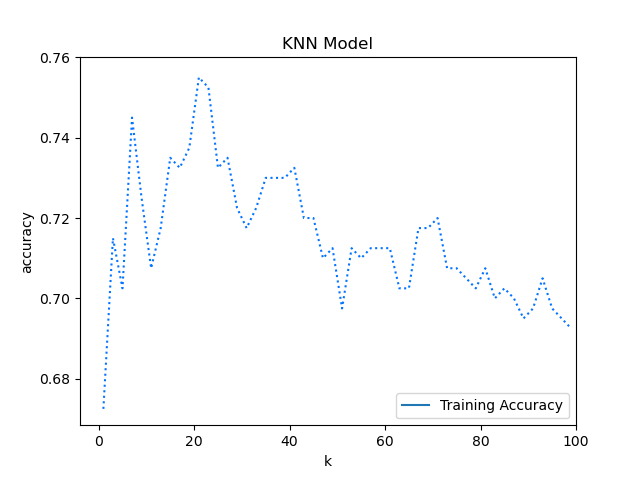
The *k*-nearest neighbor's algorithm assumes that similar things exist nearby. In other words, similar things are near to each other. the *K*-NN is a [non-parametric](https://en.wikipedia.org/wiki/Non-parametric_statistics) method proposed used for [classification](https://en.wikipedia.org/wiki/Statistical_classification) and [regression](https://en.wikipedia.org/wiki/Regression_analysis).

I implement the model with the libraries: keras, sklearn, numpy, glob, PIL, matlibplot. Skelarn contains a knn algorithm for vectors and, I used numpy to convert the images into a vector, and matlibplot to export a chart describing the learning process. Image size is 200x200 and turned into a vector of 40,000 parameters to check the distance between.

This model wasn’t that successful, it was better than I expected but still not good enough for classification. The model accuracy was only managed to get up to 75.5% accuracy on the validation set and 75% the test set for k = 21.

The algorithm runs from k = 1 to 99.

**Here are the results:**

**** Loading dataset successfully!~

Setting up arguments...

Start the knn algorithm!~

Round: 0 K is now: 1 accuracy: 0.6725

Round: 1 K is now: 3 accuracy: 0.715

Round: 2 K is now: 5 accuracy: 0.7025

Round: 3 K is now: 7 accuracy: 0.745

…

Round: 8 K is now: 17 accuracy: 0.7325

Round: 9 K is now: 19 accuracy: 0.7375

Round: 10 K is now: 21 accuracy: 0.755

…

Round: 17 K is now: 35 accuracy: 0.73

Round: 18 K is now: 37 accuracy: 0.73

Round: 19 K is now: 39 accuracy: 0.73

…

Round: 27 K is now: 55 accuracy: 0.71

Round: 28 K is now: 57 accuracy: 0.7125

Round: 29 K is now: 59 accuracy: 0.7125

…

Round: 47 K is now: 95 accuracy: 0.6975

Round: 48 K is now: 97 accuracy: 0.695

Round: 49 K is now: 99 accuracy: 0.6925

Final K = 21 accuracy = 0.755

Test Accuracy: 0.75

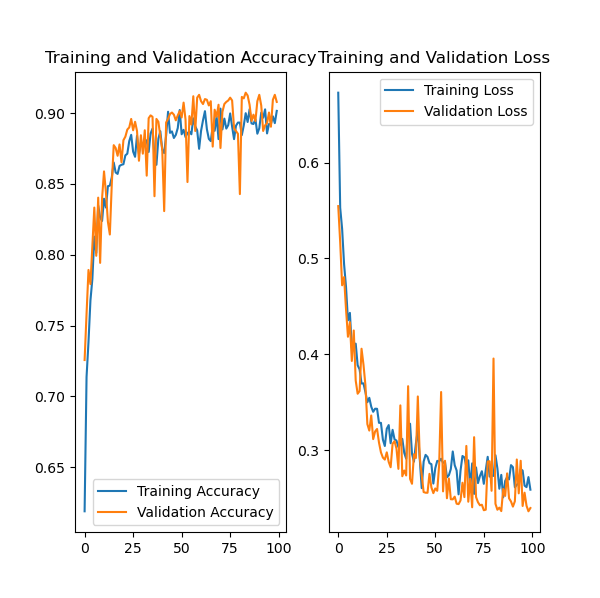
**Logistic Regression:**

Logistic regression is a function that translates the input into one of two categories, The “classic” application of logistic regression model is a binary classification. You can think of logistic regression as an on-off switch. It can stand alone, or some version of it may be used as a mathematical component to form switches, or gates, that relay or block the flow of information.

For this model, I used grayscale images and resized them into 50x50. The libraries: TensorFlow, keras, matlibplot, and JSON.

With TensorFlow and keras I built the full model. I used the originally sigmoid function. The results were better than the knn model. With 2500 weights (the number of the pixels) classified into two classes.

**Results**:

****Epoch 100/100

1/92 [..............................] - ETA: 5s - loss: 0.3897 - accuracy: 0.8148

3/92 [..............................] - ETA: 3s - loss: 0.2849 - accuracy: 0.8765

5/92 [>.............................] - ETA: 3s - loss: 0.2673 - accuracy: 0.8815

8/92 [=>............................] - ETA: 2s - loss: 0.2477 - accuracy: 0.8912

10/92 [==>...........................] - ETA: 2s - loss: 0.2568 - accuracy: 0.8907

13/92 [===>..........................] - ETA: 2s - loss: 0.2586 - accuracy: 0.8917

16/92 [====>.........................] - ETA: 2s - loss: 0.2518 - accuracy: 0.8981

19/92 [=====>........................] - ETA: 1s - loss: 0.2512 - accuracy: 0.8957

21/92 [=====>........................] - ETA: 1s - loss: 0.2433 - accuracy: 0.8986

23/92 [======>.......................] - ETA: 1s - loss: 0.2438 - accuracy: 0.8994

26/92 [=======>......................] - ETA: 1s - loss: 0.2383 - accuracy: 0.9031

29/92 [========>.....................] - ETA: 1s - loss: 0.2353 - accuracy: 0.9049

32/92 [=========>....................] - ETA: 1s - loss: 0.2396 - accuracy: 0.9034

35/92 [==========>...................] - ETA: 1s - loss: 0.2380 - accuracy: 0.9058

38/92 [===========>..................] - ETA: 1s - loss: 0.2329 - accuracy: 0.9079

41/92 [============>.................] - ETA: 1s - loss: 0.2344 - accuracy: 0.9070

44/92 [=============>................] - ETA: 1s - loss: 0.2368 - accuracy: 0.9061

47/92 [==============>...............] - ETA: 1s - loss: 0.2361 - accuracy: 0.9054

50/92 [===============>..............] - ETA: 1s - loss: 0.2395 - accuracy: 0.9052

53/92 [================>.............] - ETA: 0s - loss: 0.2421 - accuracy: 0.9029

56/92 [=================>............] - ETA: 0s - loss: 0.2463 - accuracy: 0.9018

59/92 [==================>...........] - ETA: 0s - loss: 0.2483 - accuracy: 0.8999

62/92 [===================>..........] - ETA: 0s - loss: 0.2528 - accuracy: 0.8976

65/92 [====================>.........] - ETA: 0s - loss: 0.2537 - accuracy: 0.8974

68/92 [=====================>........] - ETA: 0s - loss: 0.2529 - accuracy: 0.8987

71/92 [======================>.......] - ETA: 0s - loss: 0.2538 - accuracy: 0.8991

74/92 [=======================>......] - ETA: 0s - loss: 0.2516 - accuracy: 0.9002

77/92 [========================>.....] - ETA: 0s - loss: 0.2546 - accuracy: 0.8987

80/92 [=========================>....] - ETA: 0s - loss: 0.2532 - accuracy: 0.8993

83/92 [==========================>...] - ETA: 0s - loss: 0.2535 - accuracy: 0.8998

85/92 [==========================>...] - ETA: 0s - loss: 0.2525 - accuracy: 0.9002

88/92 [===========================>..] - ETA: 0s - loss: 0.2529 - accuracy: 0.9000

91/92 [============================>.] - ETA: 0s - loss: 0.2545 - accuracy: 0.9001

92/92 [==============================] - 3s 32ms/step - loss: 0.255 - accuracy: 0.90 | valloss: 0.2263 val accuracy: 0.9209

test loss: 0.22562633075825483 test accurecy: 0.9182513

**Multilayer Perceptron:**

A multilayer perceptron is a logistic regressor where instead of feeding the input to the logistic regression you insert a intermediate layer, called the hidden layer, that has a nonlinear activation function (usually tanh or sigmoid) . One can use many such hidden layers making the architecture deep.

This model was a bit different than the logistic regression and more complex. I found that if I add too many hidden layers to the architecture it makes the model less successful. In this specific model, I used three hidden layers with sigmoid function in every layer to prevent linear results. The image size in 50x50, and when flatten the vector is 2500. Also, I used the "adam" optimizer, and the loss is computed with "binary cross-entropy".

1st layer: 1250 weights fully connected to the image vector.

2nd layer: 512 weights fully connected to the 1st layer.

3rd layer: 10 weights fully connected to the 2nd layer.

After the 3rd layer comes to classification into two classes that also fully connected to the 3rd layer.

I have tried to insert more layers or to delete some, but no better results. At the bigging, I took the logistic regression code (the same code) and I insert a hidden layer right between the picture vector and the result layer. I didn’t notice any change with the results so I insert another hidden layer, now the result got better in 1.5%. then I tried to insert 2 more hidden layers but the results got worst by 4%, so I thought 2 layer was optimal but just to be sure I tried 3 hidden layers and the result got better in additional 0.5% so I played with the number of dots in every layer and notice that when I use 1250, 512, 10 I get the optimal results with improving 0.06%-0.03%.

**Results:**

Epoch 100/100

1/92 [..............................] - ETA: 18s - loss: 0.2216 - accuracy: 0.9259

3/92 [..............................] - ETA: 12s - loss: 0.3001 - accuracy: 0.8827

5/92 [>.............................] - ETA: 11s - loss: 0.2513 - accuracy: 0.9111

6/92 [>.............................] - ETA: 11s - loss: 0.2595 - accuracy: 0.9074

7/92 [=>............................] - ETA: 10s - loss: 0.2391 - accuracy: 0.9127

8/92 [=>............................] - ETA: 10s - loss: 0.2602 - accuracy: 0.9005

11/92 [==>...........................] - ETA: 9s - loss: 0.2570 - accuracy: 0.9091

12/92 [==>...........................] - ETA: 9s - loss: 0.2372 - accuracy: 0.9120

13/92 [===>..........................] - ETA: 9s - loss: 0.2364 - accuracy: 0.9117

16/92 [====>.........................] - ETA: 8s - loss: 0.2537 - accuracy: 0.9039

17/92 [====>.........................] - ETA: 8s - loss: 0.2534 - accuracy: 0.9020

22/92 [======>.......................] - ETA: 7s - loss: 0.2550 - accuracy: 0.9049

23/92 [======>.......................] - ETA: 7s - loss: 0.2506 - accuracy: 0.9066

25/92 [=======>......................] - ETA: 7s - loss: 0.2587 - accuracy: 0.9030

26/92 [=======>......................] - ETA: 7s - loss: 0.2651 - accuracy: 0.8996

31/92 [=========>....................] - ETA: 6s - loss: 0.2633 - accuracy: 0.8996

34/92 [==========>...................] - ETA: 6s - loss: 0.2638 - accuracy: 0.8976

39/92 [===========>..................] - ETA: 5s - loss: 0.2622 - accuracy: 0.8989

40/92 [============>.................] - ETA: 5s - loss: 0.2618 - accuracy: 0.9081

41/92 [============>.................] - ETA: 5s - loss: 0.2601 - accuracy: 0.9084

42/92 [============>.................] - ETA: 5s - loss: 0.2612 - accuracy: 0.9077

48/92 [==============>...............] - ETA: 4s - loss: 0.2557 - accuracy: 0.9082

49/92 [==============>...............] - ETA: 4s - loss: 0.2559 - accuracy: 0.9072

50/92 [===============>..............] - ETA: 4s - loss: 0.2556 - accuracy: 0.9178

51/92 [===============>..............] - ETA: 4s - loss: 0.2553 - accuracy: 0.9187

54/92 [================>.............] - ETA: 3s - loss: 0.2548 - accuracy: 0.9168

58/92 [=================>............] - ETA: 3s - loss: 0.2545 - accuracy: 0.9275

71/92 [======================>.......] - ETA: 2s - loss: 0.2510 - accuracy: 0.9112

72/92 [======================>.......] - ETA: 2s - loss: 0.2514 - accuracy: 0.9113

73/92 [======================>.......] - ETA: 1s - loss: 0.2519 - accuracy: 0.9109

74/92 [=======================>......] - ETA: 1s - loss: 0.2513 - accuracy: 0.9107

75/92 [=======================>......] - ETA: 1s - loss: 0.2506 - accuracy: 0.9105

76/92 [=======================>......] - ETA: 1s - loss: 0.2518 - accuracy: 0.9104

80/92 [=========================>....] - ETA: 1s - loss: 0.2517 - accuracy: 0.9107

81/92 [=========================>....] - ETA: 1s - loss: 0.2503 - accuracy: 0.9113

82/92 [=========================>....] - ETA: 1s - loss: 0.2496 - accuracy: 0.9116

83/92 [==========================>...] - ETA: 0s - loss: 0.2484 - accuracy: 0.9123

88/92 [===========================>..] - ETA: 0s - loss: 0.2484 - accuracy: 0.9109

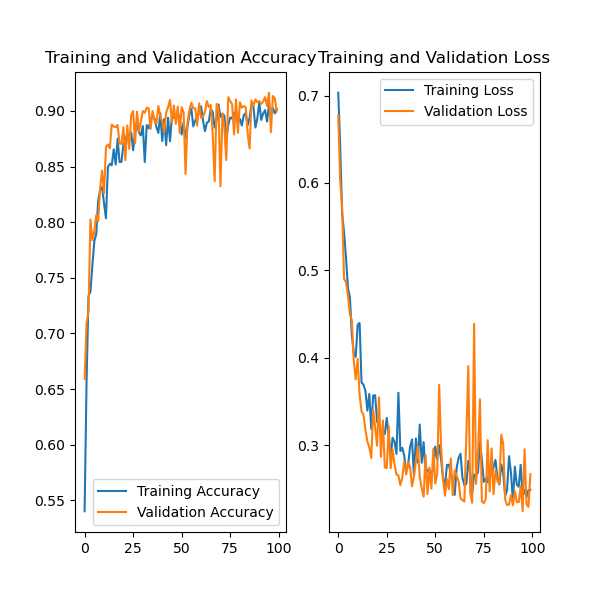
89/92 [============================>.] - ETA: 0s - loss: 0.2479 - accuracy: 0.9112

90/92 [============================>.] - ETA: 0s - loss: 0.2476 - accuracy: 0.9113

91/92 [============================>.] - ETA: 0s - loss: 0.2473 - accuracy: 0.9116

92/92 [==============================] - 11s 120ms/step - loss: 0.247 accuracy: 0.911 | val loss: 0.266 val accuracy: 0.932

test loss: 0.24428823261487823 test accuracy: 0.9301793



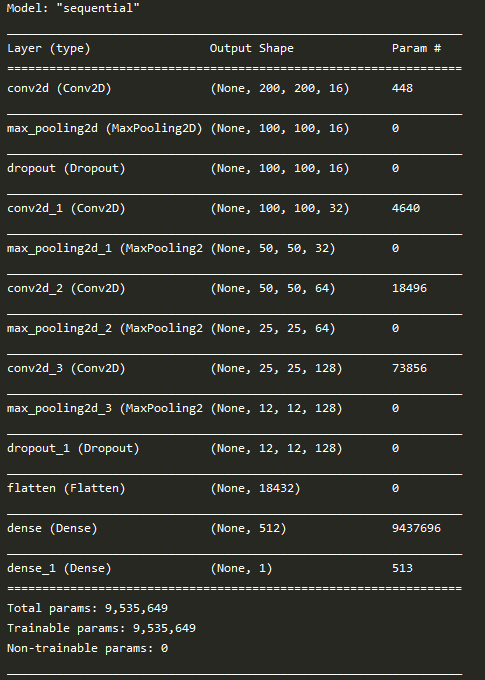
as you can see the results look very much like the results from the logistic regression model, the prediction got better only by 2%.

In conclusion This technique does not find the true capabilities of this technology, we must take it even further to allow the model to successfully predict a good and required machine learning.

**Convolutional neural network:**

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.

This technique is much complex than the other so far, in this method, I used optimization techniques, adding noise to images, "relu" activation, a large number of layers, padding, and max-pooling.

**Architecture:**

The data set for this method was colored images size 200x200.

In this technique, the prediction improved pretty well, unlike the logistic regression and the MLP, this model was able to commit real learning and to give much better results. The validation accuracy got 96% and the train got 97% accuracy after 250 epochs.

I had to try a few structures before I got these results but in general, I tried to make a structure similar to what I learned in Amos Azaria's course on neural networks.

Results:

Epoch 250/250

1/93 [..............................] - ETA: 4:16 - loss: 0.1078 - accuracy: 0.9531

2/93 [..............................] - ETA: 3:45 - loss: 0.0982 - accuracy: 0.9531

3/93 [..............................] - ETA: 3:36 - loss: 0.1075 - accuracy: 0.9531

4/93 [>.............................] - ETA: 3:29 - loss: 0.1308 - accuracy: 0.9453

5/93 [>.............................] - ETA: 3:23 - loss: 0.1184 - accuracy: 0.9500

6/93 [>.............................] - ETA: 3:19 - loss: 0.1294 - accuracy: 0.9479

7/93 [=>............................] - ETA: 3:16 - loss: 0.1227 - accuracy: 0.9531

12/93 [==>...........................] - ETA: 3:02 - loss: 0.1534 - accuracy: 0.9440

13/93 [===>..........................] - ETA: 2:59 - loss: 0.1513 - accuracy: 0.9447

16/93 [====>.........................] - ETA: 2:52 - loss: 0.1469 - accuracy: 0.9404

19/93 [=====>........................] - ETA: 2:45 - loss: 0.1383 - accuracy: 0.9441

22/93 [======>.......................] - ETA: 2:38 - loss: 0.1431 - accuracy: 0.9411

25/93 [=======>......................] - ETA: 2:31 - loss: 0.1474 - accuracy: 0.9406

28/93 [========>.....................] - ETA: 2:24 - loss: 0.1494 - accuracy: 0.9392

29/93 [========>.....................] - ETA: 2:22 - loss: 0.1483 - accuracy: 0.9402

30/93 [========>.....................] - ETA: 2:20 - loss: 0.1479 - accuracy: 0.9411

31/93 [=========>....................] - ETA: 2:17 - loss: 0.1455 - accuracy: 0.9420

35/93 [==========>...................] - ETA: 2:08 - loss: 0.1432 - accuracy: 0.9429

36/93 [==========>...................] - ETA: 2:06 - loss: 0.1416 - accuracy: 0.9440

37/93 [==========>...................] - ETA: 2:04 - loss: 0.1400 - accuracy: 0.9451

38/93 [===========>..................] - ETA: 2:01 - loss: 0.1408 - accuracy: 0.9453

41/93 [============>.................] - ETA: 1:55 - loss: 0.1437 - accuracy: 0.9444

44/93 [=============>................] - ETA: 1:48 - loss: 0.1443 - accuracy: 0.9425

47/93 [==============>...............] - ETA: 1:41 - loss: 0.1430 - accuracy: 0.9418

50/93 [===============>..............] - ETA: 1:35 - loss: 0.1400 - accuracy: 0.9428

53/93 [================>.............] - ETA: 1:28 - loss: 0.1418 - accuracy: 0.9422

56/93 [=================>............] - ETA: 1:21 - loss: 0.1458 - accuracy: 0.9408

59/93 [==================>...........] - ETA: 1:15 - loss: 0.1443 - accuracy: 0.9417

62/93 [===================>..........] - ETA: 1:08 - loss: 0.1453 - accuracy: 0.9410

65/93 [===================>..........] - ETA: 1:01 - loss: 0.1482 - accuracy: 0.9399

68/93 [====================>.........] - ETA: 55s - loss: 0.1487 - accuracy: 0.9396

71/93 [=====================>........] - ETA: 48s - loss: 0.1478 - accuracy: 0.9393

74/93 [======================>.......] - ETA: 42s - loss: 0.1464 - accuracy: 0.9398

77/93 [=======================>......] - ETA: 35s - loss: 0.1467 - accuracy: 0.9401

80/93 [========================>.....] - ETA: 28s - loss: 0.1468 - accuracy: 0.9406

83/93 [=========================>....] - ETA: 22s - loss: 0.1461 - accuracy: 0.9413

86/93 [==========================>...] - ETA: 15s - loss: 0.1460 - accuracy: 0.9419

87/93 [===========================>..] - ETA: 13s - loss: 0.1486 - accuracy: 0.9409

88/93 [===========================>..] - ETA: 11s - loss: 0.1477 - accuracy: 0.9412

89/93 [===========================>..] - ETA: 8s - loss: 0.1470 - accuracy: 0.9414

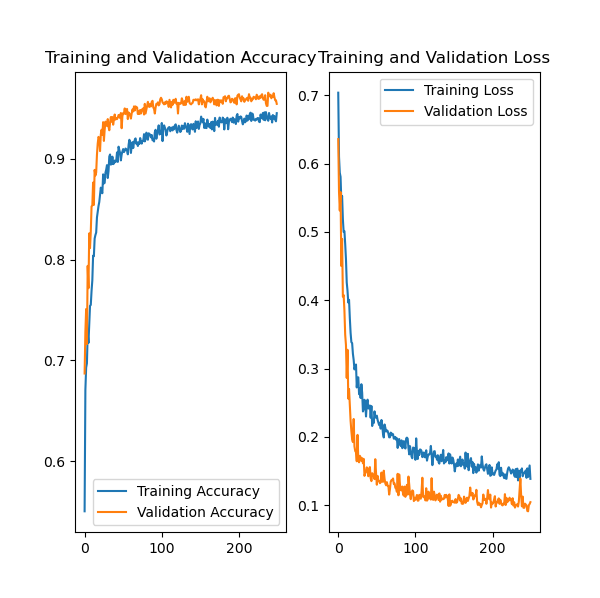
90/93 [============================>.] - ETA: 6s - loss: 0.1468 - accuracy: 0.9413

91/93 [============================>.] - ETA: 4s - loss: 0.1465 - accuracy: 0.9413

92/93 [============================>.] - ETA: 2s - loss: 0.1459 - accuracy: 0.9414

93/93 [==============================] - ETA: 0s loss: 0.1461 accuracy: 0.9414 Validation - loss: 0.0829 - accuracy: 0.9677

test loss: 0.07626534784760546 test accuracy: 0.9716685



**Personal notes:**

This project was not easy for me, also because I did it myself but in my opinion, I learned in the best way to understand the techniques learned in the course. Apart from the methods I did for the project I tried to do both AdaBoost and SVM but unfortunately the results were not at a sufficient level.

In the beginning, I tried to implement logistic regression and MLP with the help of TensorFlow 1.5 and very quickly realized that this is a difficult task that may not bear fruit, and I started to go in another direction and discovered of in TensorFlow 2.1, this library allows us to build models in a much more abstract way. And so I created these two methods without such a problem.

Indeed to create the CNN model I had a lot more complex to adapt architecture so that you would learn well and also run at a reasonable time for a personal laptop.

I used information from a course I did at the university on deep learning and natural language processing by Amos Azaria.

# The KNN model was very challenging to build it myself. I decided to build it with the help of a library called Scikit-learn that would allow me to run faster and more efficiently. To build the model I followed a guide I found online. Link to the guide [HERE](https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn?utm_source=adwords_ppc&utm_campaignid=9942305733&utm_adgroupid=100189364546&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=229765585183&utm_targetid=dsa-929501846124&utm_loc_interest_ms=&utm_loc_physical_ms=9070282&gclid=Cj0KCQjwpZT5BRCdARIsAGEX0zlt1dFjAlXcsbIldbEckWiIJwenTbOFXWD75BUGS2TBxETy1dJOc6waAr7mEALw_wcB). This model took more time in terms of work than the other models because it is different from them in terms of structure and concept.

I think this model was less successful than others because it seems to me that it is not possible to accurately predict images and categorize them only by examining distances from the nearest neighbor and determining the result as the value of the neighbor. The reason is, in this case, is that although there are differences between men and women, when it comes to just a picture of a face it is more difficult to classify pictures of women with short hair or men with long hair, men without facial hair or prominent facial bones, all these parameters become values ​​on a vector. And it can not be determined only by close distance values ​​of neighbors.

For the other models, you can certainly understand why CNN is better than the other two models. But in general, the models are better because they are much more complex models with weights that are recalculated in each round with backpropagation according to the model's need to adapt to the values ​​he was right and wrong.

**conclusion:**

After this project, my understanding of the methods grew and expanded. Undoubtedly it can be seen that the methods of deep learning are much better than different methods of machine learning.

I believe that with better hardware I could have gotten better results with CNN but the results I got show that machine learning can reach good levels of recognition, and even as good as in humans.

Thanks for reading,

**Tal Noam**