CNN FOR BREAST CANCER DETECTION

Machine learning project

Team Members:

- Esraa Khaled
- Heba Ali
- Marina Maher
- Shaimaa Gamal

• Introduction:

Breast cancer is a malignant cell growth in the breast. If left untreated, cancer spreads to other areas of the body. Excluding skin cancer, breast cancer is the most common type of cancer in women in the United States, accounting for one of every three cancer diagnoses. It is a fatal disease, with about 35% of women in Egypt are diagnosed with it, and about 42 thousand deaths happen yearly around the world, which made it the interest of scientists and researchers over long decades. Some researchers headed towards specific chemicals used to kill it and others towards various treatment devices like a linear accelerator. Biomedical engineers are headed towards using engineering concepts like machine learning tools to diagnose it, which we will do in this project.

● Data used:

We used the popular **BreakHis dataset** [1] with 39545 images, 12400 benign -fig1 a-, and 27145 malignant -fig1 b-. It can be noticed that the data is unbalanced. This could be a problem as one class can dominate the other while training and the model won't learn it enough and stuck with it during testing. This has an easy solution in python which is stratified sampling discussed in preprocessing section.

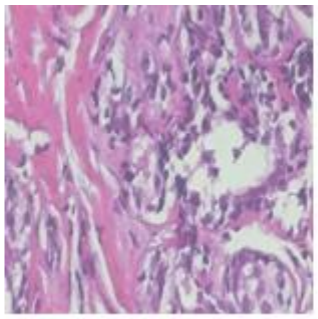


fig1-a benign

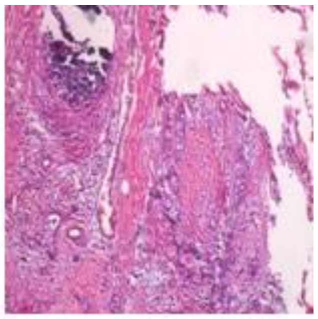


fig1-b malignant

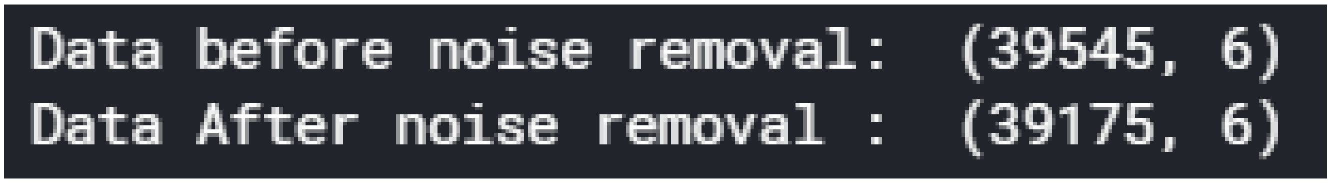
• Preprocessing:

Steps done:

1. Noise removal:

We have our images in BGR format with (n, m, 3) dimensions in python. Most of our images shape is (460, 700, 3). We started by flattening them, which means converting them into a 1D array of size n*m*3 and in our case: 460 * 700 * 3 = 966,000. After flattening, we removed the small number of images with flattening size! =966,000 because they are noisy data in our model. Each image has an amount of information or in other words noise. This amount of noise differs by the change in the image size. If we pass an image of a certain size to our CNN model, it adapts to this noise by applying filters to the image. Imagine passing many images of the same size to the model while training it, and all of a sudden it sees a few images of different sizes. The model didn't have enough time to know them, so they are considered as noisy data, which causes low accuracy [2].

Result of this step:



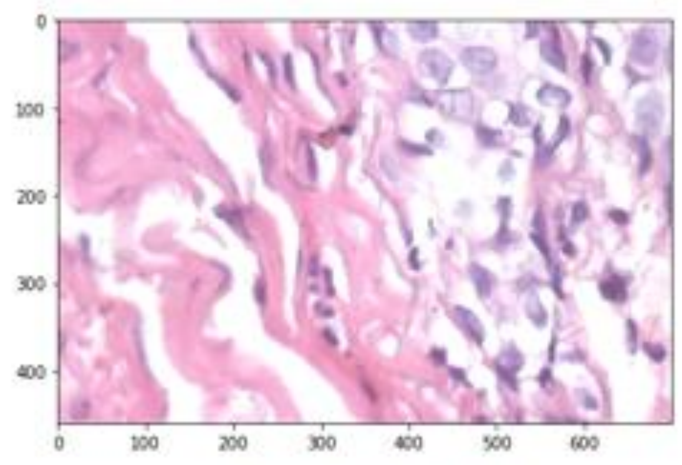
```
Data before noise removal:  (39545, 6)
Data After noise removal :  (39175, 6)
```

2. Resizing and Color conversion:

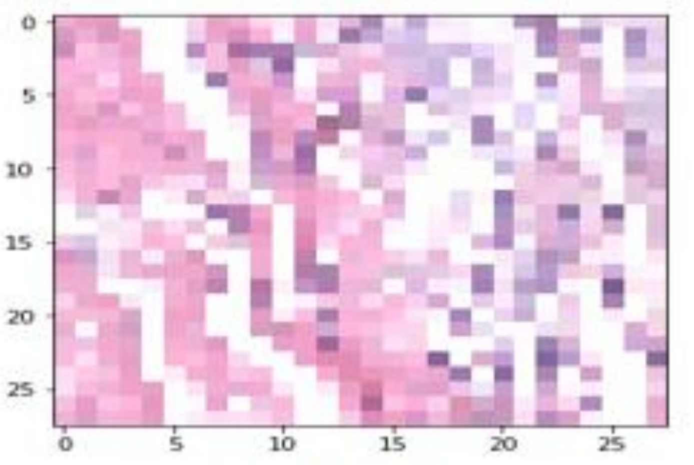
We resized all of our images to (28,28,3) to fasten the model, which lowers the resolution but decreased the running time. We tried running on the actual size of (460, 700, 3), but it took much time in the first epochs and then fails due to memory error as it needs high processing machines with high GPU.

We had done color conversion twice, first, we converted all the images to RGB to ensure that they are in the needed format and the second conversion was to HSV as to the best of our knowledge this is the commonly used color space to detect the different region of interest in biomedical images.

Result of this step:



Size= (460,700,3)

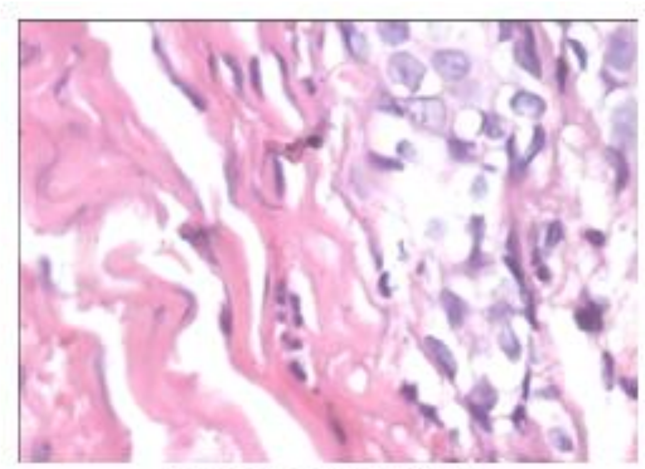


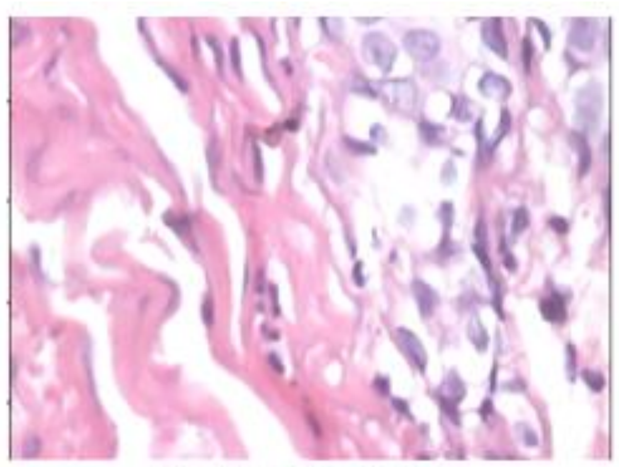
Size= (28,28,3)

3. **<u>Median Blurring:</u>**

We applied cv2.medianBlur() function to remove salt-and-pepper noise, this is considered as high-frequency content by applying a low-pass filter. It computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value.

Result of this step:

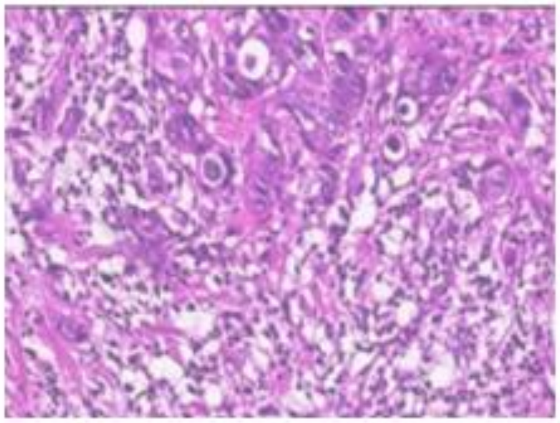

before median blurring

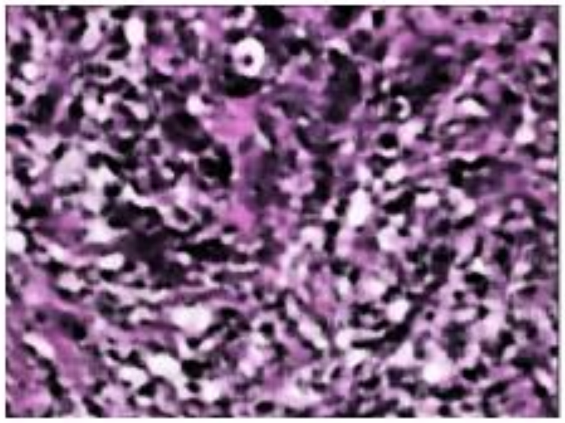

after median blurring

4. Histogram Equalization:

It is a method that improves the contrast in an image, in order to stretch out the intensity range as seen in the following image applied on our data.

Result of this step:

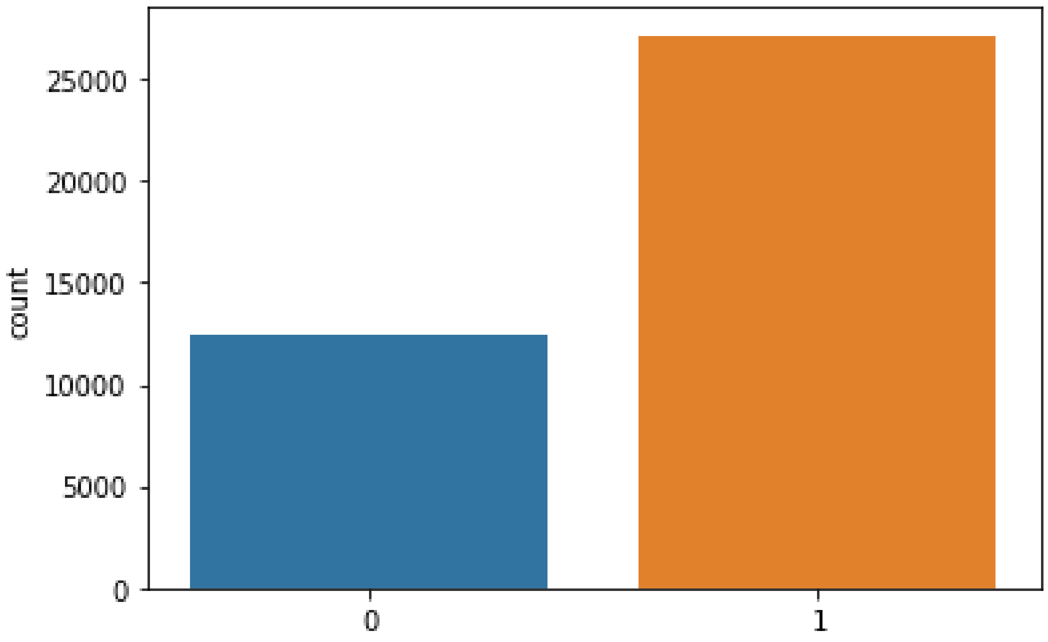

Before Histogram Equalization


After Histogram Equalization

5. Stratified sampling:

As discussed in the data section, our data is unbalanced. By applying stratified sampling, we ensure that the number of benign and malignant images are the same in the training and testing sets. This ensures that the model is trained and tested well without being penalized by one class.

Visualization of difference between two classes:



• Model:

The model used is Convolution Neural Networks (CNN), a neural network with hidden layers as convolution layers. CNN is also called a feature extractor, which apply several filters on the input image to detect patterns in the image, making it a competitor in the image classification field due to its high accuracy. In other words, what CNN does is extracting the feature of the input image and converting it into a lower dimension without losing its characteristics [2]. It can use various types of filters to detect lines, edges, circles, squares, complex shapes, and any shape you can dream of – fig2-. Once you get the right filtration matrix, you can train the model, and you will have your dream classification model. These reasons are why we choose CNN in our task – benign & malignant images classification, but there is another reason which is to the best of our knowledge, makes CNN the outstanding model in this field -tumor detection- which is the filter size. Filter size if chosen to be (3,3) for example, this means that you constructed a

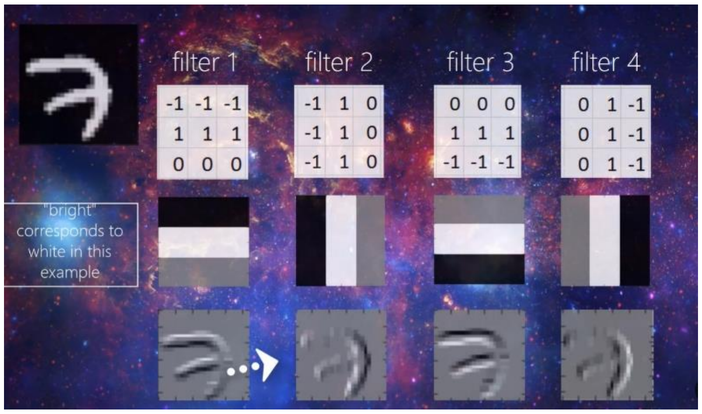


fig-2- some simple filters

3*3 matrix revolving around the tiny details of the image detecting the tumor even if its size is small. These reasons make us choose CNN to be our model, and our thoughts didn’t fail as it gives excellent accuracy.

• Our model in detail:

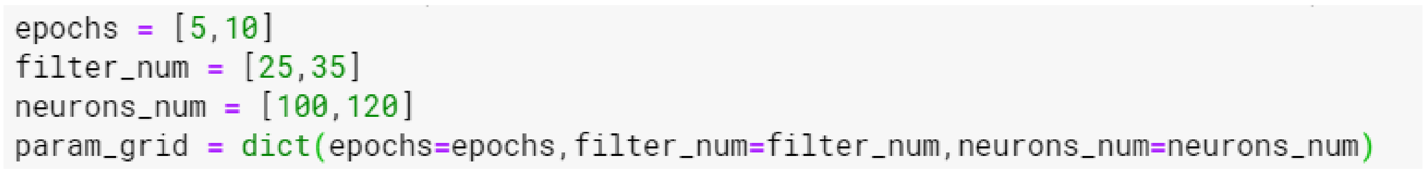
These are the parameters used, and how we choose them will be discussed later:

1. The number of filters: we used 30 filters.
2. Filter size: (3,3), as discussed earlier.
3. Strides: number of steps the filter should move after finishing its first convolution, we choose 1.
4. Padding: Padding is a process of adding zeros to the input matrix symmetrically, to maintain the dimension of output as input. You can choose “valid” -> no padding, or “same” -> padding. we tried both of them, and the first case gives higher accuracy.
5. Activation: ReLU activation to make all negative values to zero.

This was the convolution layer. After it, we put pooling layer to reduce the volume of the input image after convolution, then a flatten layer to flatten the output in preparation for the fully connected layers that make a classification decision, then a hidden layer of 150 neurons, and the output layer with two neurons indicating malignant or benign.

• Grid search CV:

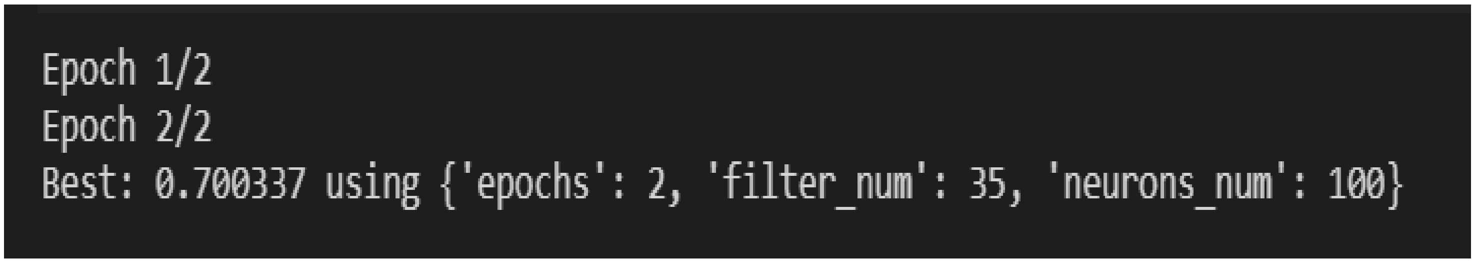
Due to a large number of hyperparameters, choosing which one to add with which one is a difficult task. So, we used grid search CV-fig3-, in which you give the values of the hyperparameters you which to try, and it begins training the model with each combination of these values and retrieves the best combination in the “ best_score_ ” parameter.



```
epochs = [5,10]
filter_num = [25,35]
neurons_num = [100,120]
param_grid = dict(epochs=epochs,filter_num=filter_num,neurons_num=neurons_num)
```

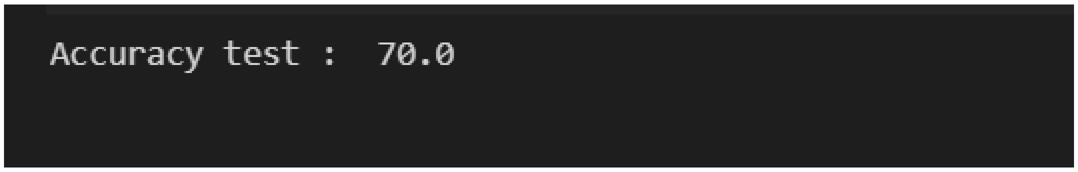
Fig-3- hyper parameters given to grid search CV

Best parameters chosen on 1500 data to run faster:



```
Epoch 1/2
Epoch 2/2
Best: 0.700337 using {'epochs': 2, 'filter_num': 35, 'neurons_num': 100}
```

Testing accuracy on this size and with grid search CV:

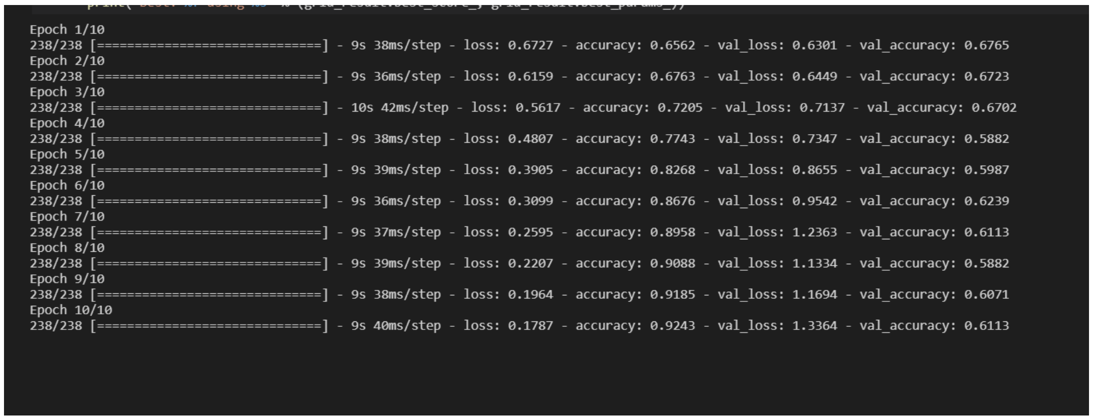


```
Accuracy test :  70.0
```

• Results:

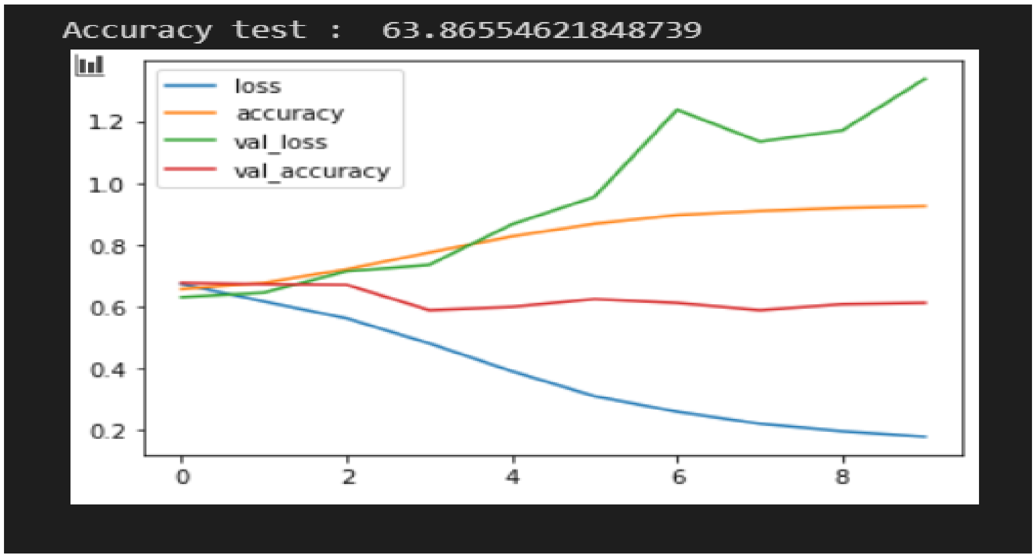
The following is the results on many samples of our data increased bit by bit on best parameters of Grid search, but on 10 epochs to see their effect more:

1. On 3000 size of our data:
   a. Accuracy report: showing how accuracy and loss varies with training:



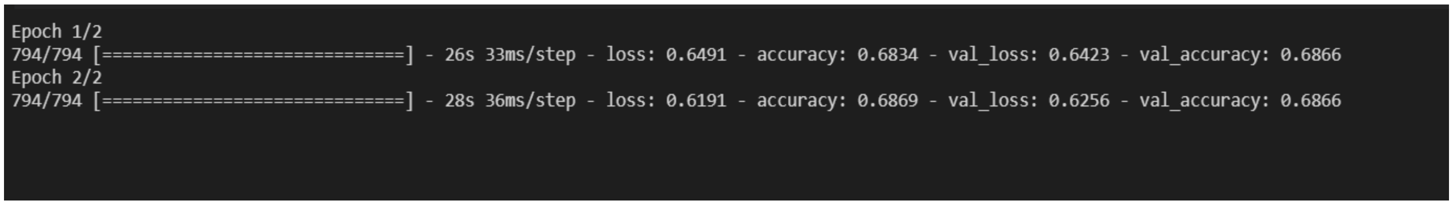
```
Epoch 1/10
238/238 [==============================] - 9s 38ms/step - loss: 0.6727 - accuracy: 0.6562 - val_loss: 0.6301 - val_accuracy: 0.6765
Epoch 2/10
238/238 [==============================] - 9s 36ms/step - loss: 0.6159 - accuracy: 0.6763 - val_loss: 0.6449 - val_accuracy: 0.6723
Epoch 3/10
238/238 [==============================] - 10s 42ms/step - loss: 0.5617 - accuracy: 0.7205 - val_loss: 0.7137 - val_accuracy: 0.6702
Epoch 4/10
238/238 [==============================] - 9s 38ms/step - loss: 0.4807 - accuracy: 0.7743 - val_loss: 0.7347 - val_accuracy: 0.5882
Epoch 5/10
238/238 [==============================] - 9s 39ms/step - loss: 0.3905 - accuracy: 0.8268 - val_loss: 0.8655 - val_accuracy: 0.5987
Epoch 6/10
238/238 [==============================] - 9s 36ms/step - loss: 0.3099 - accuracy: 0.8676 - val_loss: 0.9542 - val_accuracy: 0.6239
Epoch 7/10
238/238 [==============================] - 9s 37ms/step - loss: 0.2595 - accuracy: 0.8958 - val_loss: 1.2363 - val_accuracy: 0.6113
Epoch 8/10
238/238 [==============================] - 9s 39ms/step - loss: 0.2207 - accuracy: 0.9088 - val_loss: 1.1334 - val_accuracy: 0.5882
Epoch 9/10
238/238 [==============================] - 9s 38ms/step - loss: 0.1964 - accuracy: 0.9185 - val_loss: 1.1694 - val_accuracy: 0.6071
Epoch 10/10
238/238 [==============================] - 9s 40ms/step - loss: 0.1787 - accuracy: 0.9243 - val_loss: 1.3364 - val_accuracy: 0.6113
```

   b. Drawing how accuracy and loss varies with training and validation accuracy:



```
Accuracy test :  63.86554621848739
```

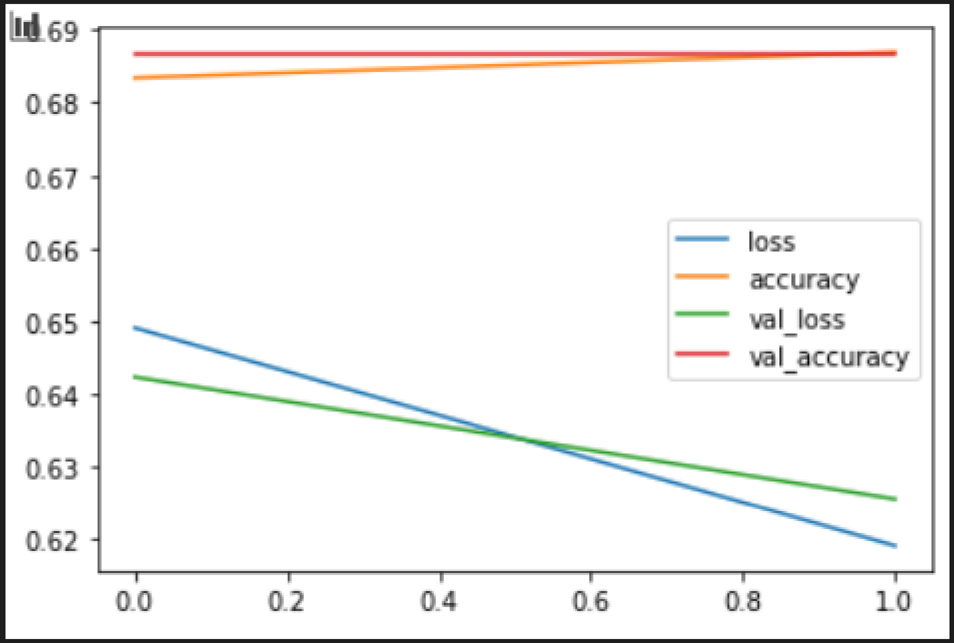
2. On 10,000 size of our data but on 2 epochs only as GridsearchCv and processing time:
    a. Accuracy report: showing how accuracy and loss varies with training:



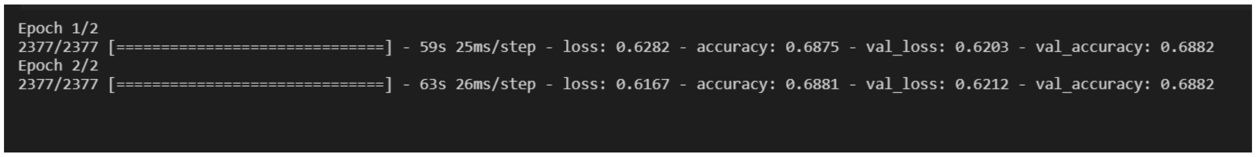
```
Epoch 1/2
794/794 [==============================] - 26s 33ms/step - loss: 0.6491 - accuracy: 0.6834 - val_loss: 0.6423 - val_accuracy: 0.6866
Epoch 2/2
794/794 [==============================] - 28s 36ms/step - loss: 0.6191 - accuracy: 0.6869 - val_loss: 0.6256 - val_accuracy: 0.6866
```

    b. Drawing how accuracy and loss varies with training and validation accuracy:

```
Accuracy test :  68.51385390428212
```

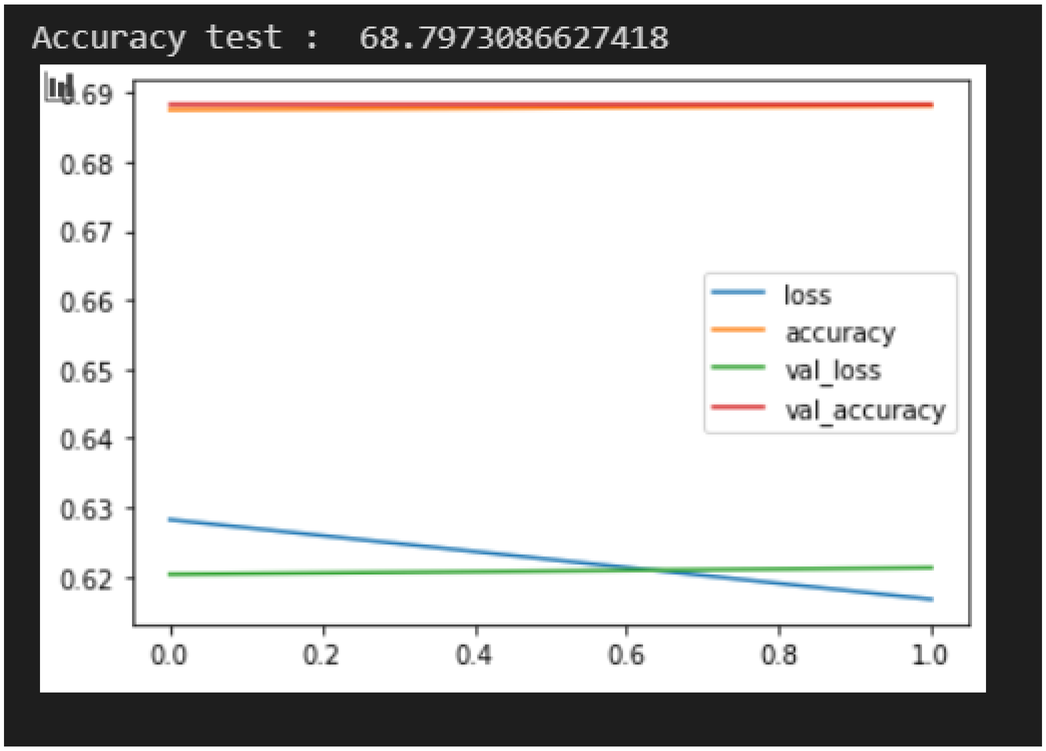


3. On 30,000 size of our data but on 2 epochs only as GridsearchCv and processing time:
   a. Accuracy report: showing how accuracy and loss varies with training:



```
Epoch 1/2
2377/2377 [==============================] - 59s 25ms/step - loss: 0.6282 - accuracy: 0.6875 - val_loss: 0.6203 - val_accuracy: 0.6882
Epoch 2/2
2377/2377 [==============================] - 63s 26ms/step - loss: 0.6167 - accuracy: 0.6881 - val_loss: 0.6212 - val_accuracy: 0.6882
```

   b. Drawing how accuracy and loss varies with training and validation accuracy:



```
Accuracy test :  68.7973086627418
```

• Problem faced:

1. Due to the massive size of data **(3.99 GB)** and the high image resolution **(460,700,3)**, this needs high processing machine which we haven't. We begin testing the model on parts of the data each step we add, but this takes a lot of time and gives low accuracy. We tried running on **500** only out of **39,545** and this took about **45 minute**s with **65% accuracy.**

2. Preprocessing steps needed before CNN is a little bit dummy on the internet with few resources, and most of the resources are about PCA as a feature extractor and SVM as a model, but we preferred to use CNN due to the mentioned reasons before, so we took a long time in our preprocessing research and we found good kernels on Kaggle to help us.

3. The long steps we used for preprocessing with the huge data took us about **five** hours to give us a **memory error** in the end, and we were forced to run it on small data of size **1500** which gives us low accuracy.

• References:

[1] BreakHis dataset

[2] Noise removal

[2] CNN