**Breast Cancer Diagnosis Using Deep Learning Algorithm**

Researcher: Maor Karakukli

Institute: Holon Institute Of Technology

Department: Electronics & Electricity Engineering

ID:203281803

**Introduction: 1.**

**Name:** Breast Cancer Diagnosis Using Deep-Learning Algorithm

**Issue**: In the clinic, medical imaging interpretation has been performed mostly by human experts such as radiologist and physicians, And we want to save time and refrain from using human sources by using computers.

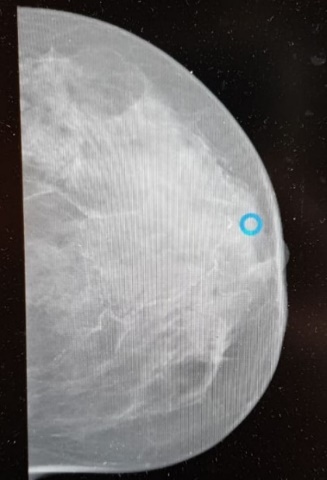
**Dataset**:INbreast-411 images(mammographys)

-implement by dicom fies,(\*\*\*.dcm) .

I choose to use 328 of them as training,41 as validation test and 41 as testing (80,10,10).

I attached 2 images for example,1 for negative and 1 for 1 for positive.

Negative Positive



So how do we tell the computer who is sick and who is healthy? **BI-RADS classification!!!**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | **Category** |
| **Known biopsy-proven malignancy – Appropriate action should be take**n | **Highly suggestive of malignancy** – Appropriate action should be taken | **Suspicious** abnormality – Biopsy should be considered | **Probably benign** | **Benign (non-cancerous**) | **Negative** | **Incomplete -** Additional imaging evaluation and/or comparison to prior mammograms is needed. | **Definition/ What it means** |

**Note:** I decided to use just healthy or sick for classification.

**Expectation** :Right prediction of 80-90% accuracy.

**2.Model Implantation/training and Architecture.**

After observation on old's researchers and Academic articles in the same issue, and after A lot of Experiments that I've done, I will introduce my Architecture/model:

Time for 1 run/instance.(Training):10 minutes.

Test:1 minute.

Paramaters: 188,514.

IDE: Google colab(GPU).

Loss function: binary\_crossentropy

Optimzer: SGD-Stochastic gradient descent

Activation function: Sigmoid

Learning rate:0.01

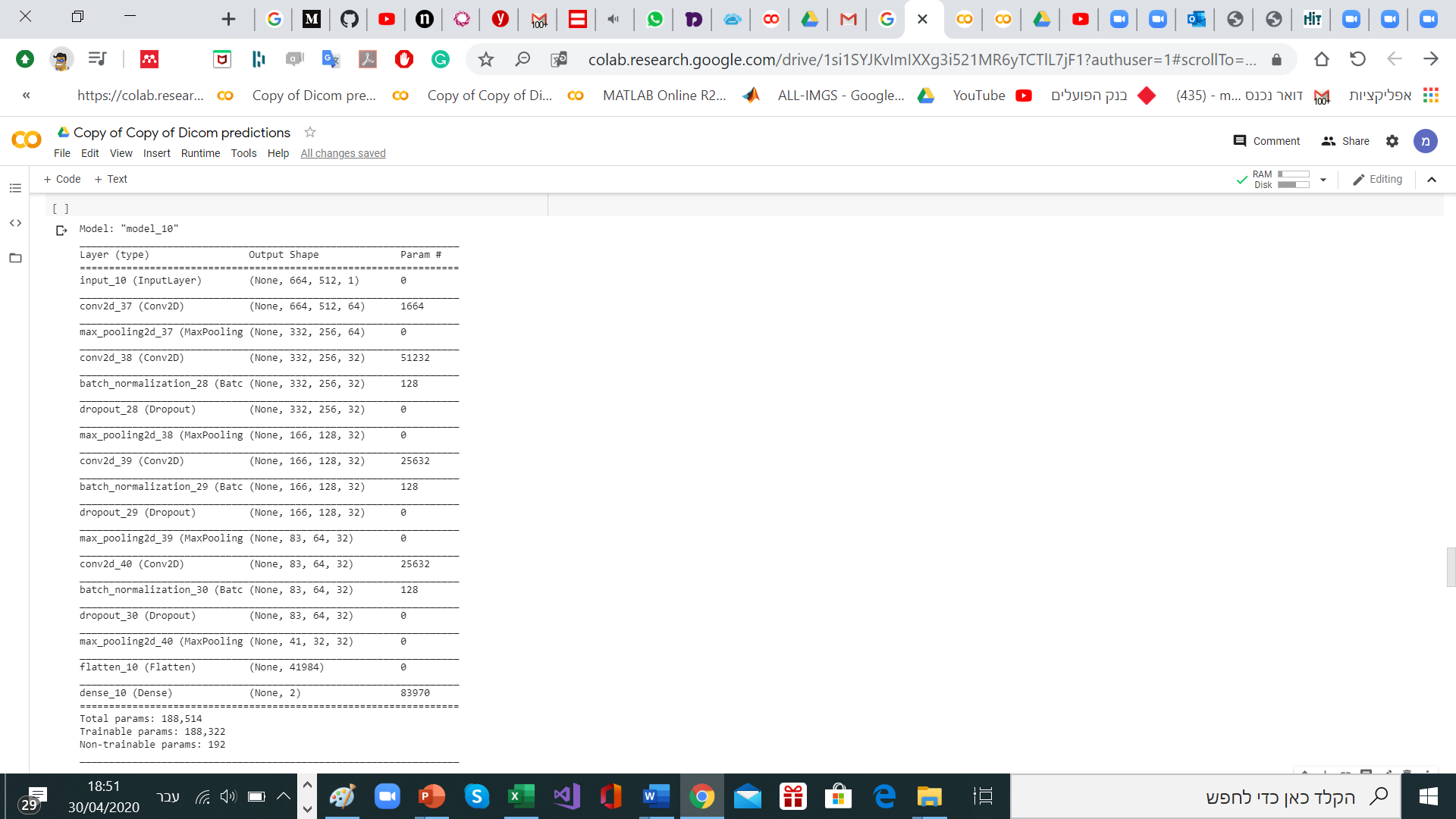
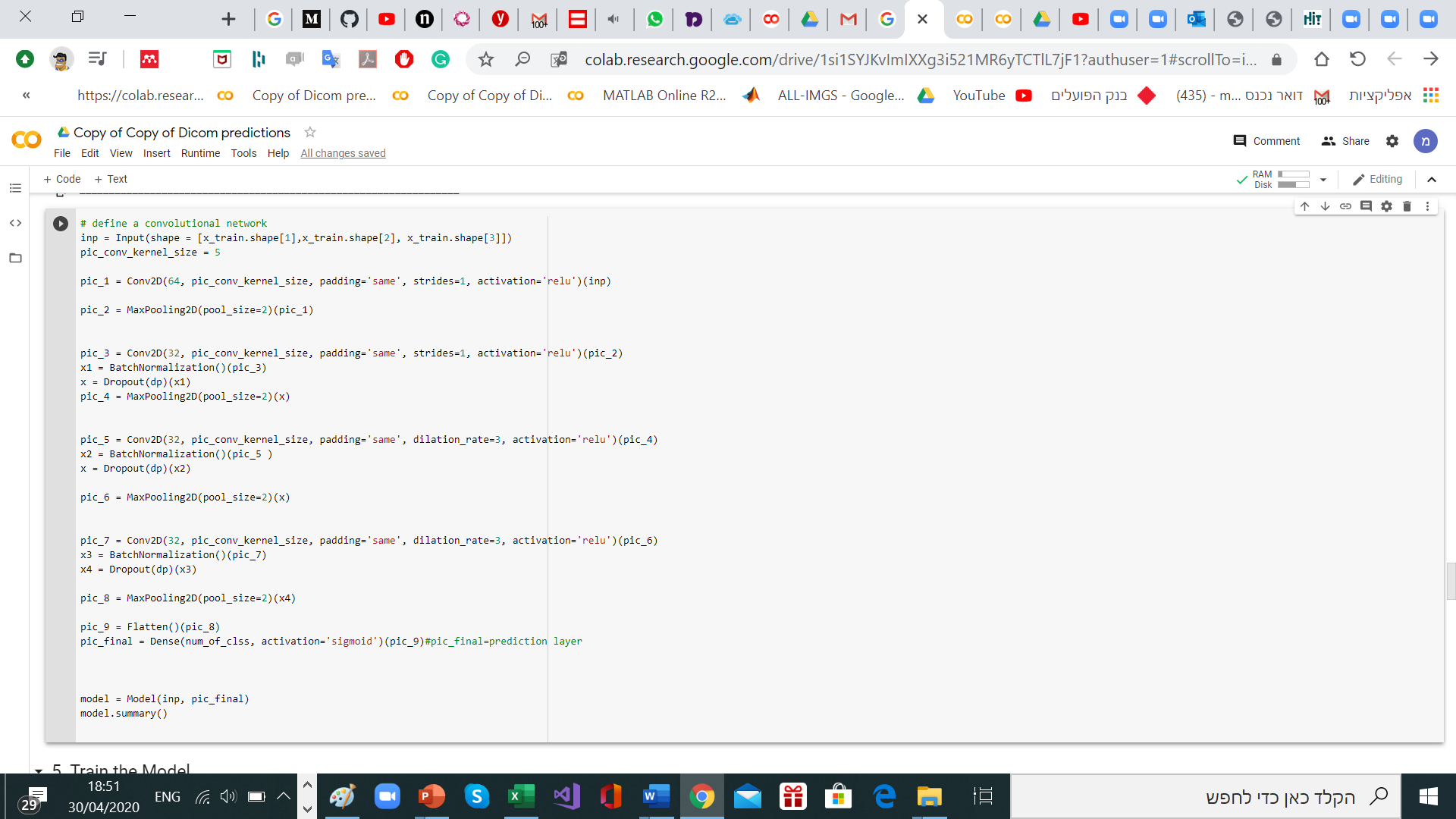
Batch Size:16.

Number of layers:17(include first layer and F.C.)

Type of layers/architecture:Conv2d-(Kernels of 64,32,32-and Relu as activation ),Dropout,maxpooling,flatten

Avoid Overfitting: I use Dropout with  dropout probability=0.6,and Early stopping with minimum delta=1e-4, and patience/Epochs=5,and flatten as F.C.

****



As we can see I use Conv2D layers,with maxpooling,and Dropout,I found it usful,after reaserchers done by me

I tryied a lot of models,including using "Adam" as optimzor,

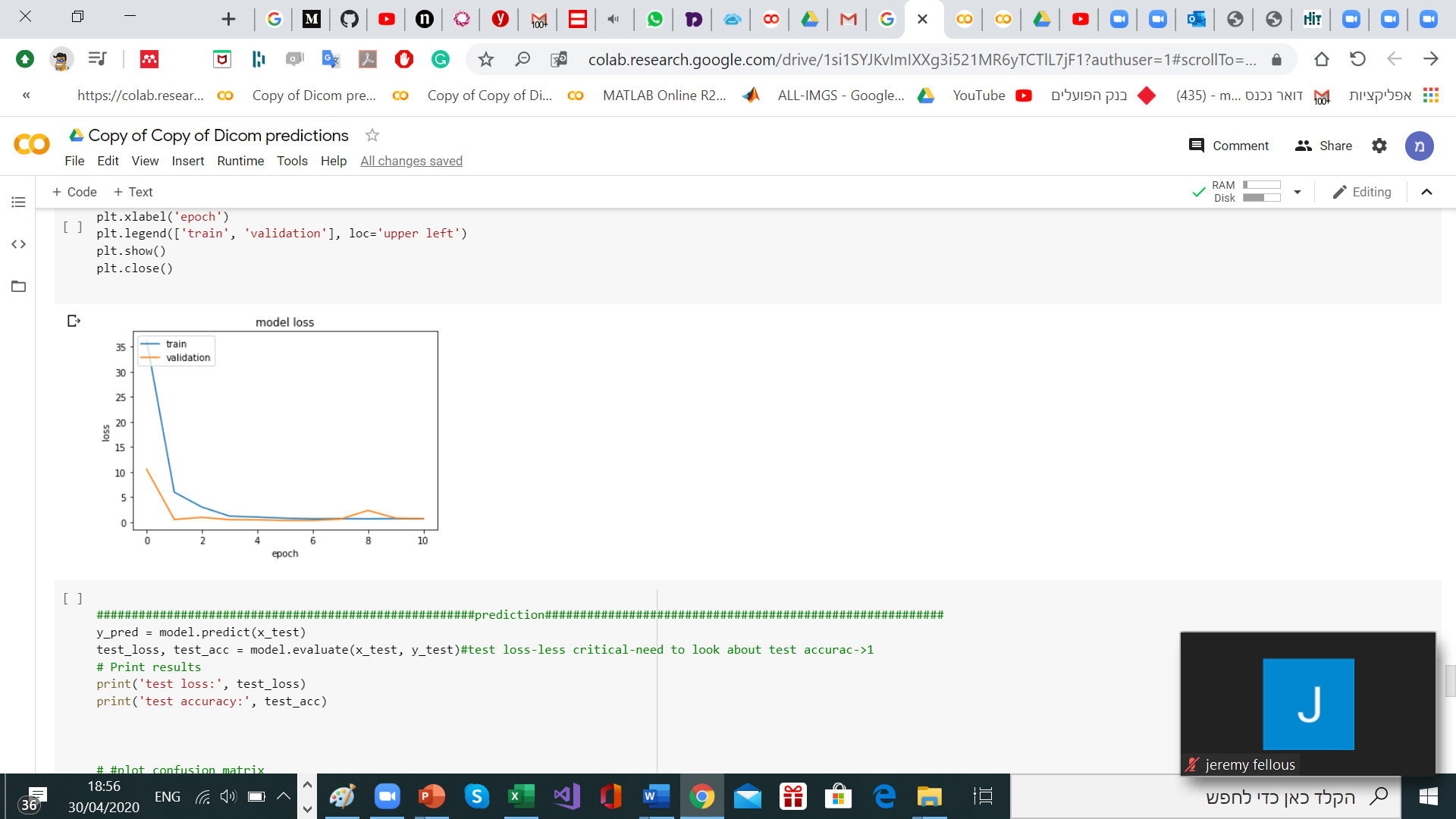
And using dense and flatten in the model,and also "softmax" as Activation(F.C layer),

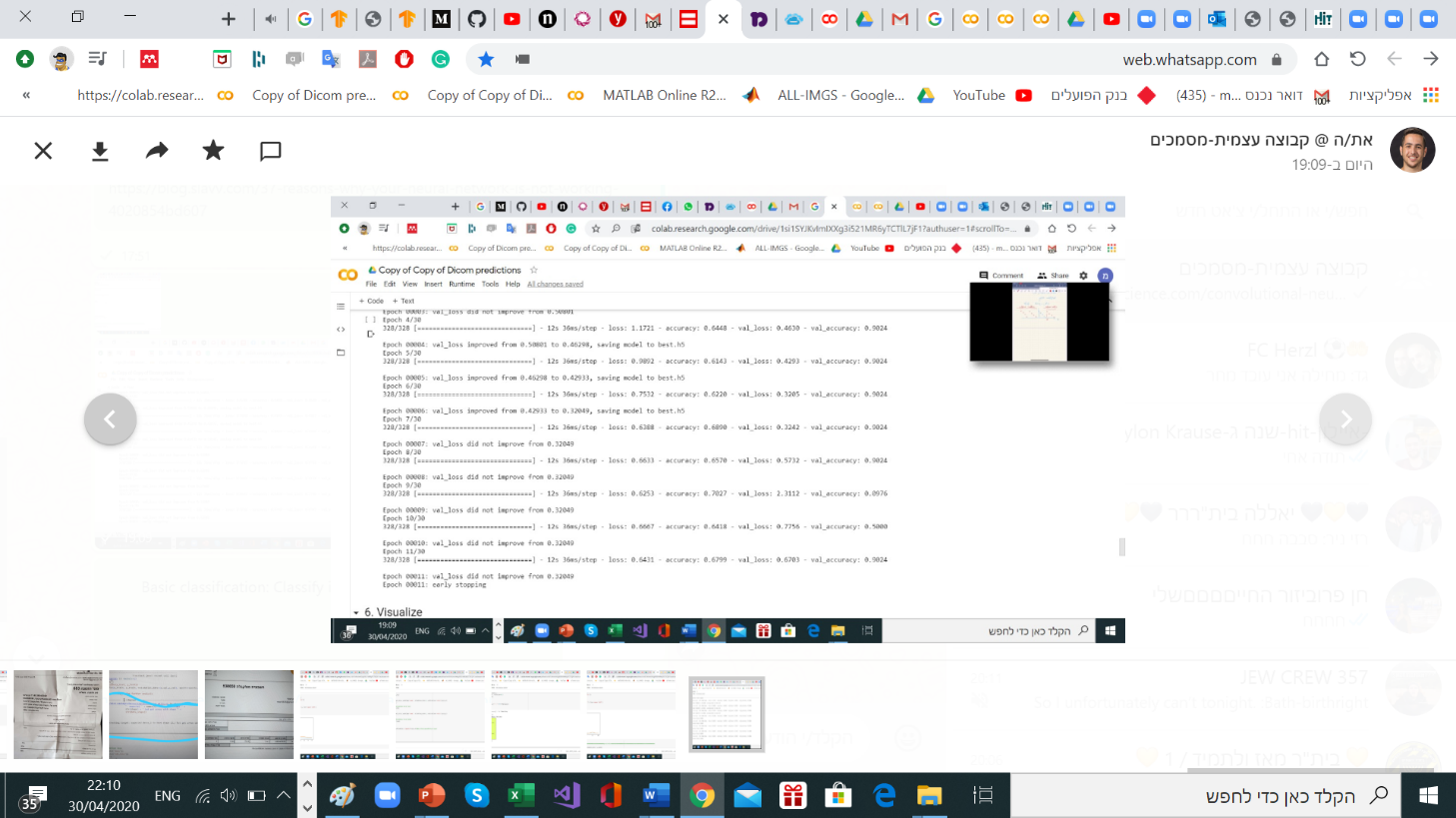
And etc.….but it was leading to bad results.

Summary of the Model:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameters** | **Output Shape** | **Type** | **Layer Name** |
| 0 |  |  | Input layer |
| 1664 | (None, 664, 512, 64) | Dense | Convolutional Layer |
| 51232 | (None, 332, 256, 32) | Dense | Hidden Layer |
| 83970 | (None, 2) | Dense | Fully connected Layer |
|  |  | Sigmoid | Activation Function |
|  |  | 20 | Numner of Epochs |
| 188,514 | | Params: | Total |
| 188,322 | | Params: | Trainable |
| 192 | | Params: | Non-trainable |

**Training results**

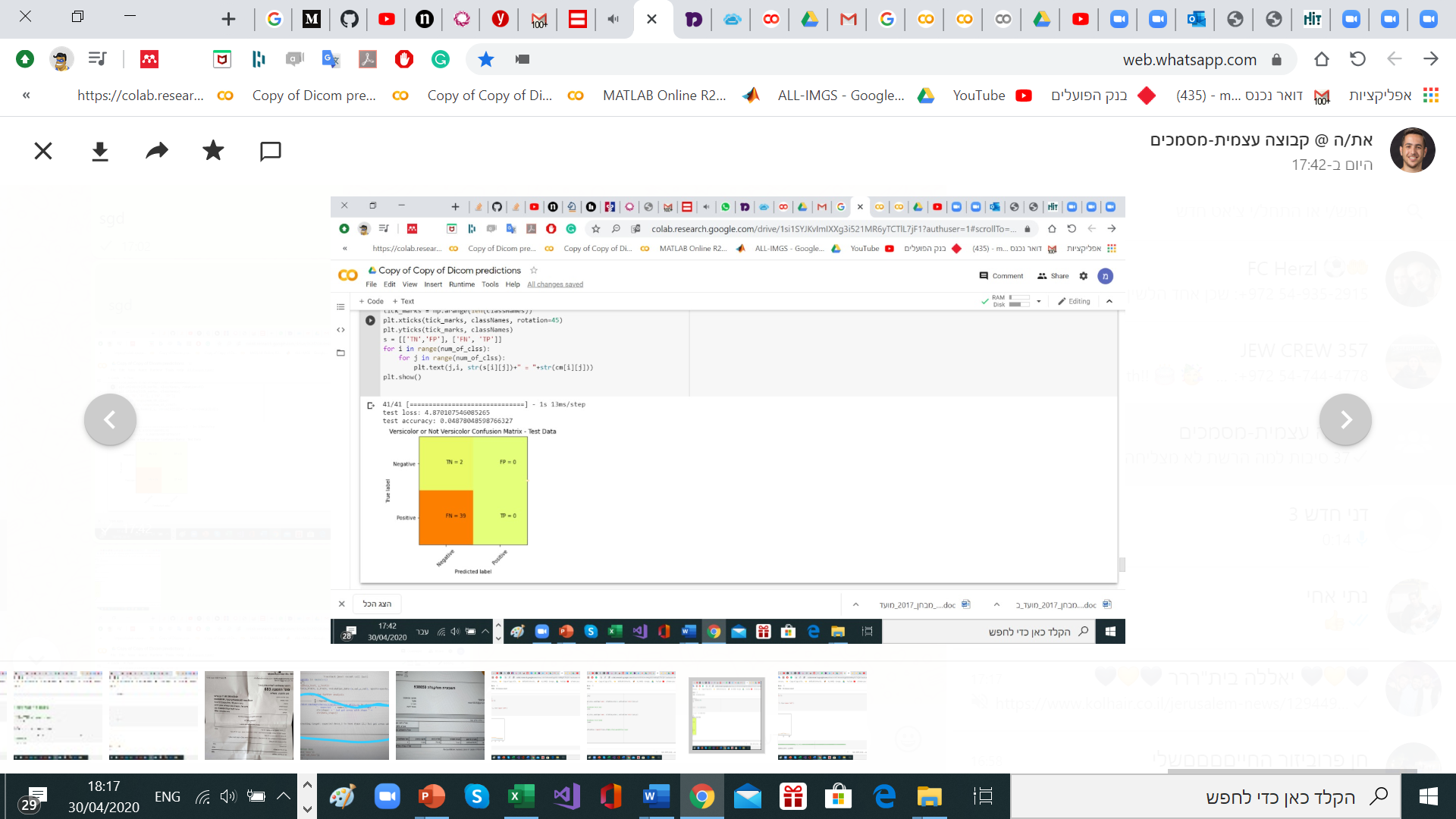




As we can see there are a good results for Training and validation.

**3).Conclusion:**

Model architecture results/Prediction:

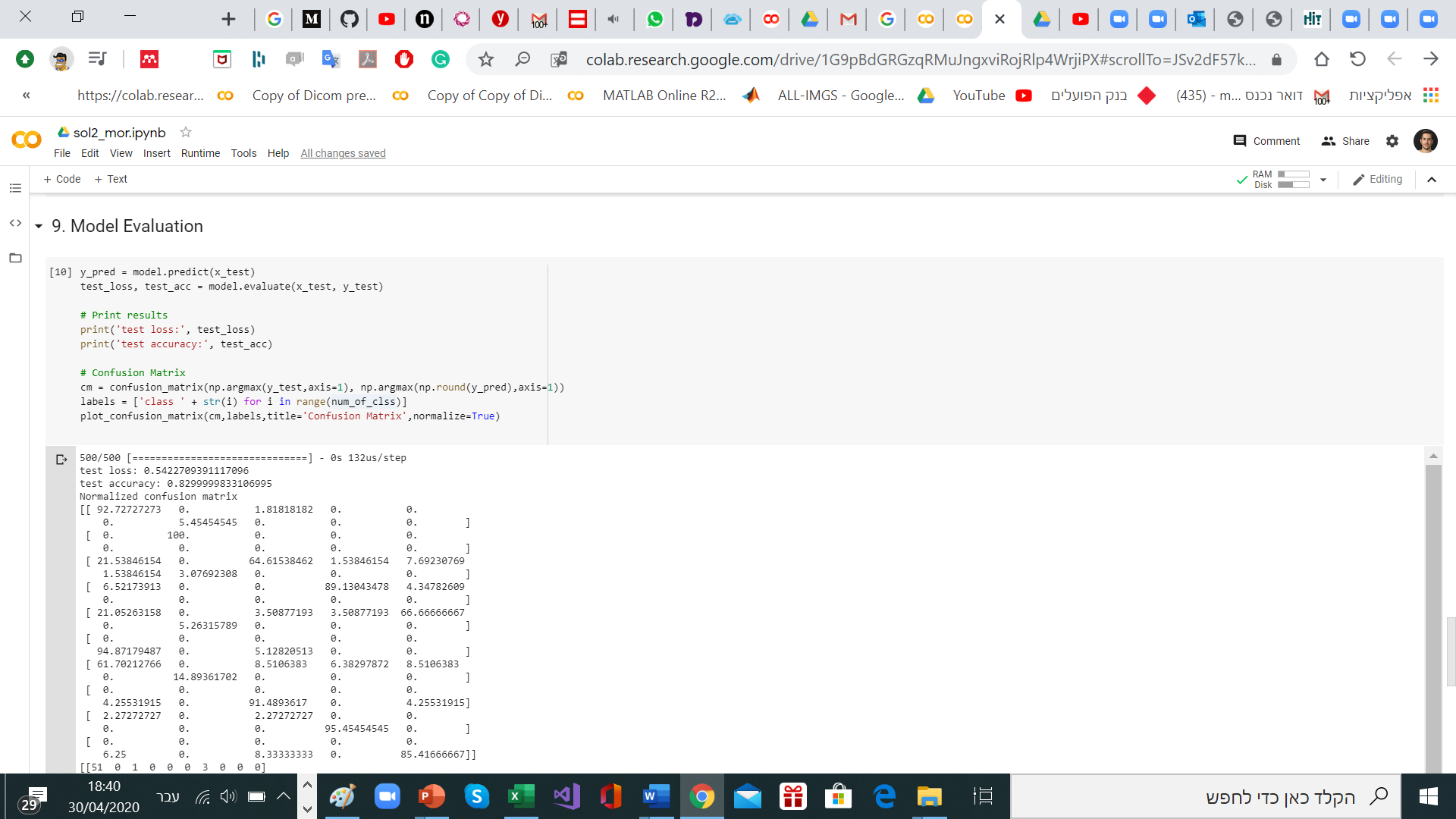


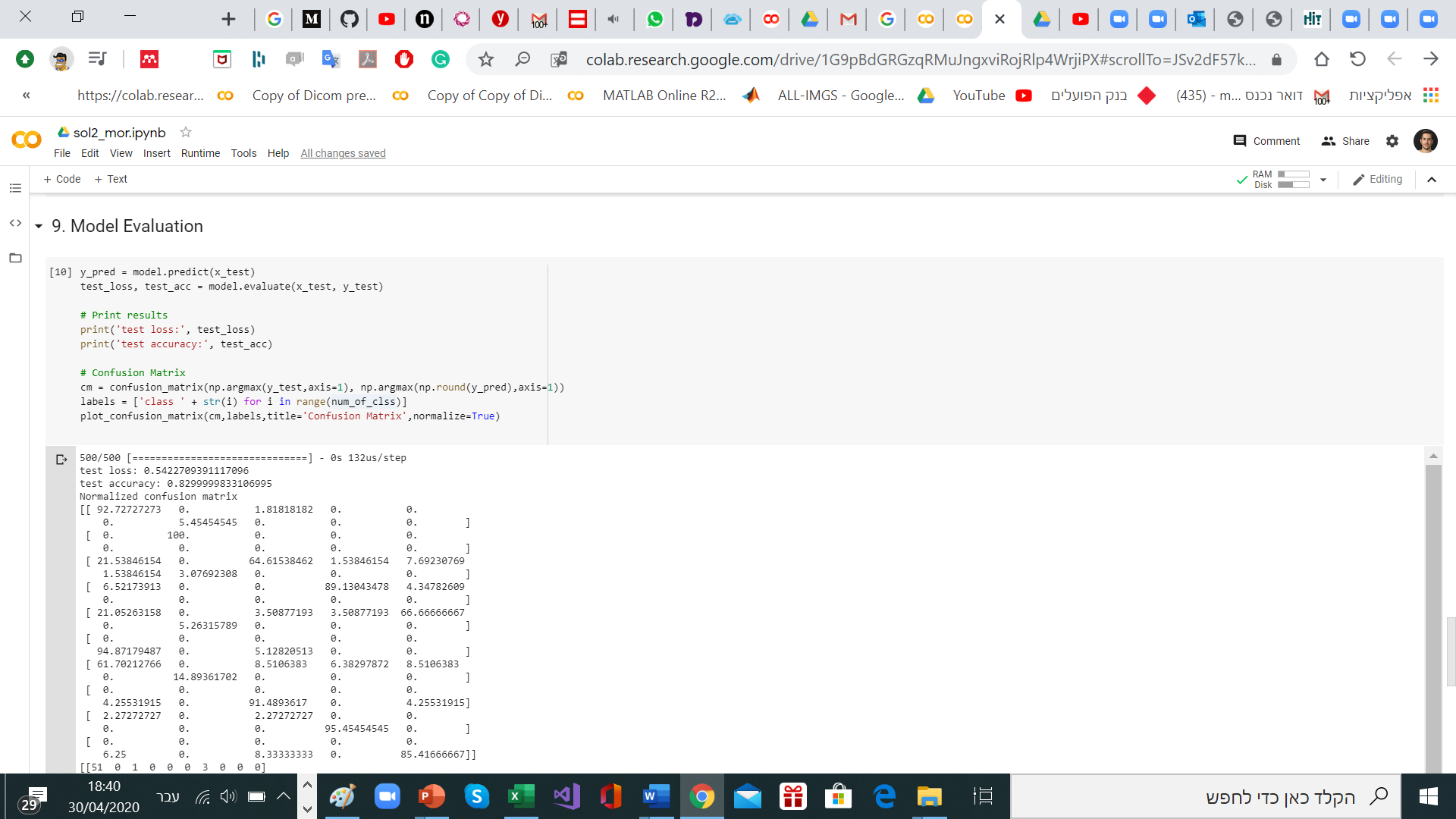
As we can see from the confusion matrix blow & from the "test accuracy", we get bad result, and the model **Failed**.

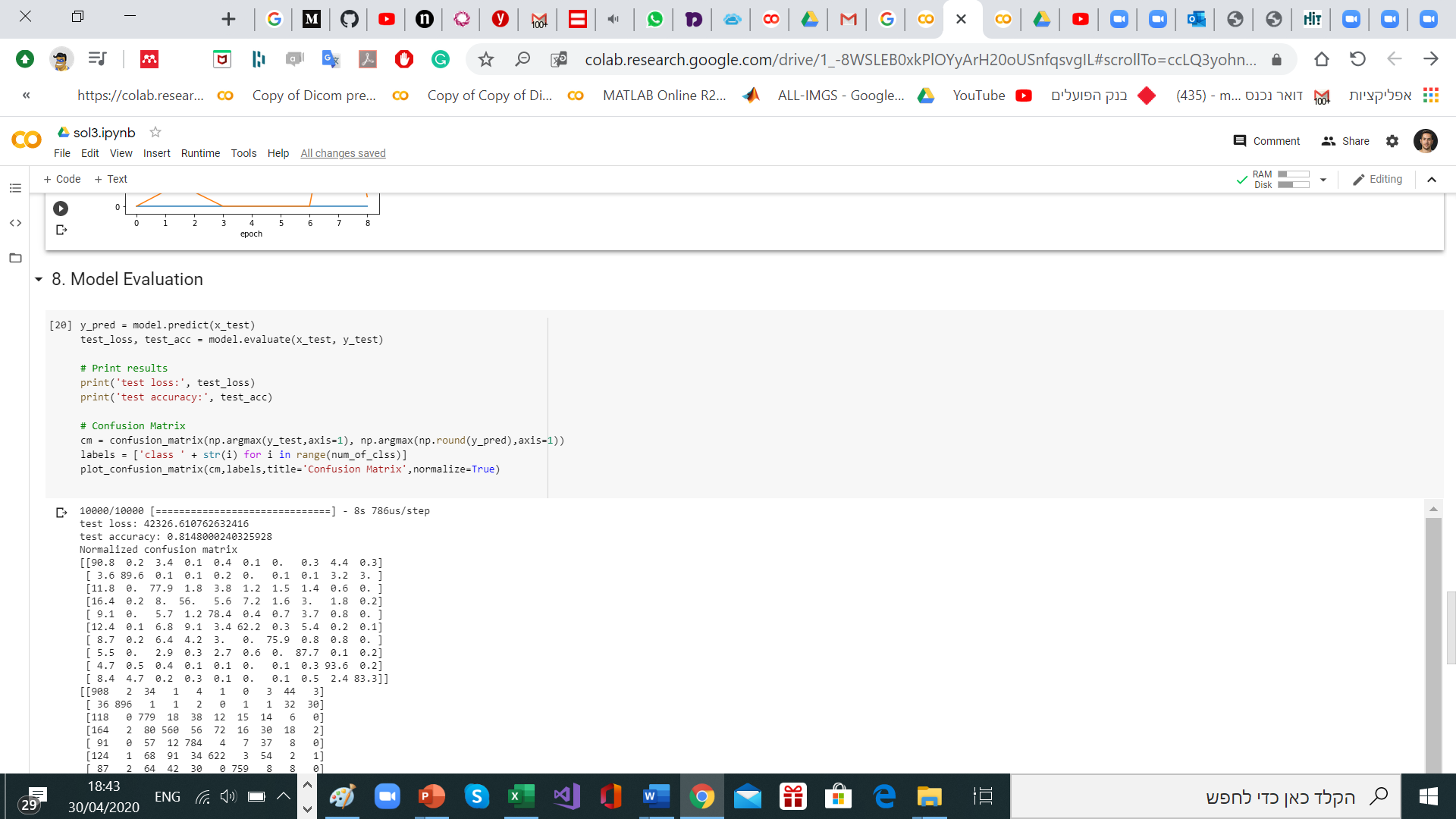
It may happen from a couple of reasons, lack of data-Despite to Cifar-10/FashionMnist we got just 411 images, and that not sufficient data to get prediction as we aspects,and for my opinion(and according to old Researches that I saw), the major and most important thing for Medical-Images DL models Is the "pre-processing" stage ,Image-processing actions need to be done here, to get good prediction.

also we can see that the prediction is for 1 class, and not for 2 as we expected, need to be checked technically.

Examples for Good result according to classics models with sufficient data:

Example no.1-FashioNmnist(1500 images):

Example no.2-Cifar-10(5000 images)



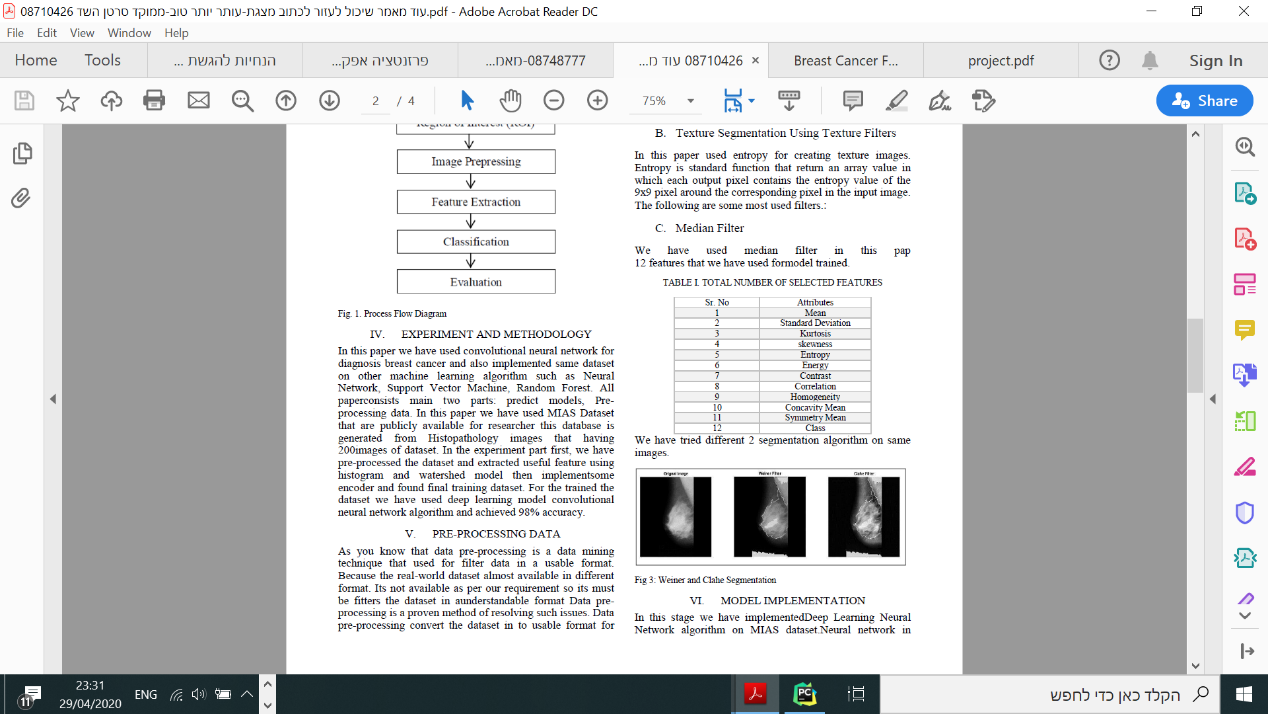
Important: There are methods to overcome that issues, For example: augmentation, Fine-Tuning and etc.

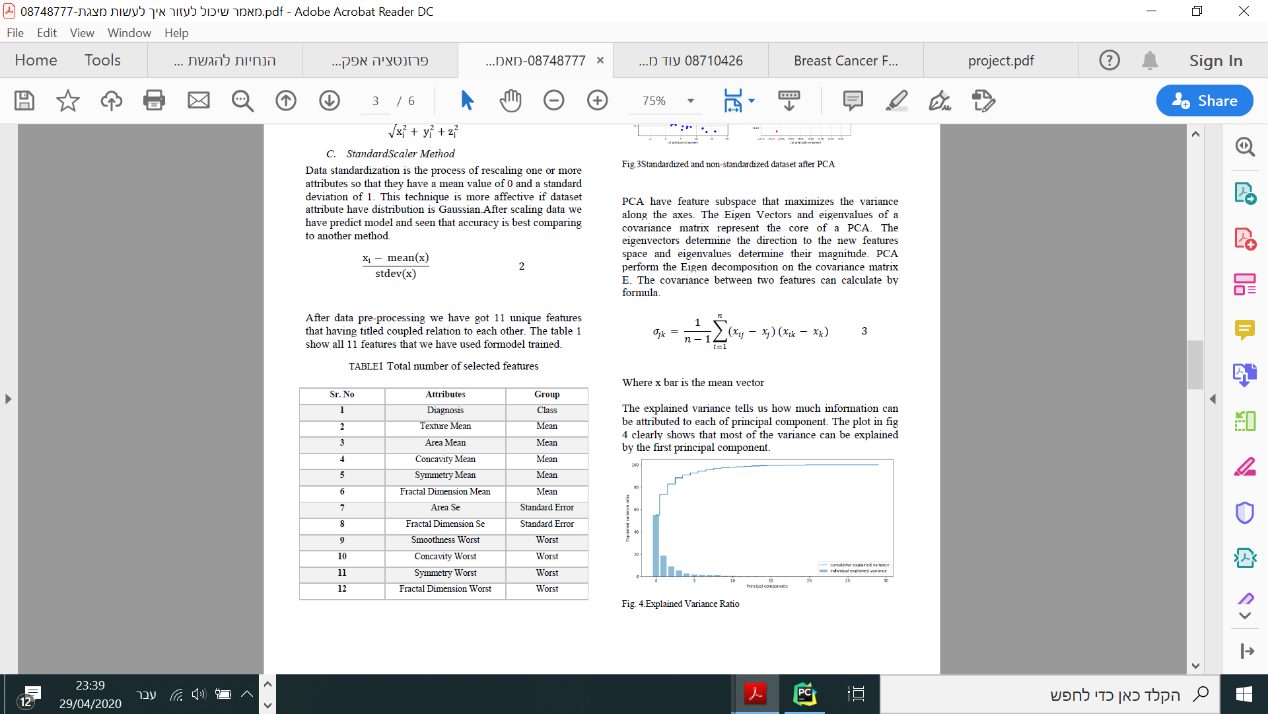
Although I didn’t use them according to Time constraints.

Researcher issue/constraint:

First,As I mention above, according to others articles, we can say that, the main issue that will advanced to good results is the "pre-processing" stage, using Image-processing techniques, and because of lack of time circumstances, we got bad result.

Examples for Pre-Processing in other data-sets:





Second, one issue that took me a bit of time is the technical think, Organization of the Data to fit the DL classic model (code issues, converting issues & Resize issues, and etc.)

to be trained susscefully. (Unlike Cifar 10/Fmnist and so on).

Solutions in the code below.

I'm pretty sure that if I had more time for training, I would get better results.

Third, since my local computer doesn’t have the right calculating powers to process the data (GPU) so I needed to use Google cloud (Colab) and that way may be good for studying but definitely **not** for developing.

To overcome those obstacles(those I success to overcome) I will work a lot with the course staff, and of course a I'll do a lot of research on the web (Facebook, Social communities of DL/ML, consulting with python programmers and Data analysts ) and in books.

**Code:**

*## 1. Import Packages  
"""  
  
# dicom installation*!pip install dicom  
!pip install pydicom  
  
*# Commented out IPython magic to ensure Python compatibility.***import** numpy **as** np  
**import** pandas **as** pd  
**import** dicom  
**import** random  
**from** keras.models **import** Sequential, Model  
**import** keras.layers **as** layers  
**from** keras.utils **import** to\_categorical  
**from** keras **import** optimizers  
**from** keras.optimizers **import** SGD  
**from** keras.models **import** model\_from\_json  
**from** keras.layers **import** Input  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.datasets **import** load\_iris  
**from** tqdm **import** tqdm  
**from** google.colab **import** files  
  
**from** sklearn.metrics **import** confusion\_matrix  
**from** skimage.transform **import** resize  
**import** re  
**from** keras.applications.vgg16 **import** VGG16, preprocess\_input  
**from** keras.callbacks **import** EarlyStopping, ModelCheckpoint  
**from** tensorflow **import** keras  
**from** keras.models **import** Sequential  
**from** keras.layers **import** Dense, Flatten, Conv2D, Input, MaxPooling2D, BatchNormalization,Dropout  
**from** keras.layers **import** Flatten  
  
*# to paint loaded dicom images for debug:***from** PIL **import** Image  
**import** numpy **as** np  
**from** matplotlib.pyplot **import** imshow  
*# %matplotlib inline***"""## 2. Load Data"""***#load images(Dicom files from drive)***import** glob  
**from** google.colab **import** drive  
drive.mount(**'/content/drive'**)  
root\_path = **'drive/My Drive/INBreast Dataset/'**images = glob.glob(root\_path + **'/ALL-IMGS/'**+ **'/\*.dcm'**)  
  
*#verify the images was loaded***print**(images)  
len(images)  
  
*#load Excel Data-set file*excel\_data = pd.read\_excel(**"drive/My Drive/INBreast Dataset/INbreast.xls"**,0,encoding=**'cp1252'**,  
 converters={**'File Name'**:str})  
  
*#validating Excel-was loaded***print**(excel\_data.head())  
**print**(excel\_data.tail())  
**print**(excel\_data.columns)  
  
*#New Data-set-convert dicmImages to list of matrixes*data\_set\_x\_train = []  
**for** img\_file **in** tqdm(images):  
 ds = dicom.read\_file(img\_file)  
 data\_set\_x\_train.append(ds.pixel\_array)  
  
*# shuffle the training data (in the same manner everytime)*random.Random(1331).shuffle(data\_set\_x\_train)  
  
*# Verify Images become a list of numpy arrays***print**(len(data\_set\_x\_train))  
**print**(data\_set\_x\_train[0].shape)  
  
*#Binary Labales,x>=4-->sick=0,x<4-->Healty=1  
# extract targets  
#list of labales/integers  
#Pandas tolist() methods is used to convert a series to list.*labels = excel\_data[**'Bi-Rads'**].tolist()  
*#list of images\_numbers/strings*files = excel\_data[**'File Name'**].tolist()  
*# 4a/4b/4c -> 4 + convert list of integers to list of strings*y\_train\_temp\_1 = list(re.findall(**r'[0-9]+'**,str(labels)))  
*#convert list of strings to list of integers*y\_train\_temp = [0 **if** int(info) >= 4 **else** 1 **for** info **in** y\_train\_temp\_1]  
*#match between labels and images,manually -create of dictionary  
#zip function Join two tuples/list together-labels and images\_names-join to dictionary*file\_to\_target = {fname: target **for** fname, target **in** zip(files, y\_train\_temp)}  
*#validation of binary labels***print**(file\_to\_target)  
len(file\_to\_target)  
  
*# format it to match data\_set\_x\_train  
#match between the dictionary(labales&images) and to the images*data\_set\_y\_train = []  
**for** filename **in** images:  
 **for** fkey **in** file\_to\_target.keys():  
 **if** fkey **in** filename:  
 data\_set\_y\_train.append(file\_to\_target[fkey])  
 **break***# print/inspect one from loaded images (and targetclasses)-final matching verifing*imgfile, image, target = images[0], data\_set\_x\_train[0], data\_set\_y\_train[0]  
**print**(imgfile)*#print image in dicom list***print**(image.shape)*#verify size of image***print**(target)*#verify label of image*imshow(image)*#plot image for example  
  
#reshape to all images-to be all be with the same size  
  
# look at image sizes-before reshapining*shapes\_temp = [img.shape **for** img **in** data\_set\_x\_train]  
**print**(set(shapes\_temp))  
*#set function shuffle the list/dictionary and print a couple of the objects randomly   
# since the shapes are different, lets convert it to one shape*data\_set\_x\_train = [resize(img, (664, 512)) **if** img.shape[0] > 664 **else** img **for** img **in** tqdm(data\_set\_x\_train)]  
**print**(data\_set\_x\_train[0].shape)  
imshow(data\_set\_x\_train[0])  
  
*#verify all shapes now match-after reshapinig*shapes = [img.shape **for** img **in** data\_set\_x\_train]  
**print**(set(shapes))  
  
*# turn all lists into np matrices*all\_train\_x\_arr = np.expand\_dims(np.stack(data\_set\_x\_train, axis=0), -1)  
all\_train\_y\_arr = np.expand\_dims(np.stack(data\_set\_y\_train, axis=0), -1)  
**print**(all\_train\_x\_arr.shape)  
**print**(all\_train\_y\_arr.shape)  
  
*# split the data to train and validation and testing->80,10,10*total = len(all\_train\_y\_arr)  
split = int(total\*0.8)  
split2=int(total\*0.9)  
  
x\_train = all\_train\_x\_arr[:split]  
y\_train = all\_train\_y\_arr[:split]  
  
  
x\_val = all\_train\_x\_arr[split:split2]  
y\_val = all\_train\_y\_arr[split:split2]  
x\_test = all\_train\_x\_arr[split2:]  
y\_test = all\_train\_y\_arr[split2:]  
  
*# Change labels to one-hot encoding*x\_train= to\_categorical(x\_train)  
y\_train = to\_categorical(y\_train)  
y\_test =to\_categorical(y\_test)  
  
x\_val = to\_categorical(x\_val)  
y\_val =to\_categorical(y\_val)  
  
**print**(**'x\_train shape:'**,np.shape(x\_train))  
**print**(**'x\_test shape:'**, np.shape(x\_test))  
**print**(**'y\_train shape:'**,np.shape(y\_train))  
**print**(**'y\_test shape:'**, np.shape(y\_test))  
  
**"""## 3. Define Parameters"""**epochs = 20 *# number of epochs*bs = 16 *# batch size*num\_of\_clss = 2 *# number of classes*lr =0.01 *# learning rate*dp = 0.6 *# dropout probability***"""## 4. Build Network"""***# define a convolutional network*inp = Input(shape = [x\_train.shape[1],x\_train.shape[2], x\_train.shape[3]])  
pic\_conv\_kernel\_size = 5  
  
*# x = Flatten()(last)  
# x = Dense(256, activation='relu')(x)  
  
# # First conv block*pic\_1 = Conv2D(64, pic\_conv\_kernel\_size, padding=**'same'**, strides=1, activation=**'relu'**)(inp)  
*#x = BatchNormalization()(pic\_1)-doesn't-work on first block*pic\_2 = MaxPooling2D(pool\_size=2)(pic\_1)  
  
*# # Second conv block  
# x = Dense(256, activation='relu')(x)  
# x = Flatten()(last)*pic\_3 = Conv2D(32, pic\_conv\_kernel\_size, padding=**'same'**, strides=1, activation=**'relu'**)(pic\_2)  
x1 = BatchNormalization()(pic\_3)  
x = Dropout(dp)(x1)  
pic\_4 = MaxPooling2D(pool\_size=2)(x)   
  
*# # Third conv block*pic\_5 = Conv2D(32, pic\_conv\_kernel\_size, padding=**'same'**, dilation\_rate=3, activation=**'relu'**)(pic\_4)  
x2 = BatchNormalization()(pic\_5 )  
x = Dropout(dp)(x2)  
*#z = Dense(y\_train.shape[-1], activation='relu')(x)*pic\_6 = MaxPooling2D(pool\_size=2)(x)  
  
*# # 4th conv block*pic\_7 = Conv2D(32, pic\_conv\_kernel\_size, padding=**'same'**, dilation\_rate=3, activation=**'relu'**)(pic\_6)  
x3 = BatchNormalization()(pic\_7)  
x4 = Dropout(dp)(x3)  
*#y = Dense(y\_train.shape[-1], activation='relu')(x4)*pic\_8 = MaxPooling2D(pool\_size=2)(x4)  
  
*#Fully connected*pic\_9 = Flatten()(pic\_8)  
pic\_final = Dense(num\_of\_clss, activation=**'sigmoid'**)(pic\_9)*#pic\_final=prediction layer*model = Model(inp, pic\_final)  
model.summary()  
  
**"""## 5. Train the Model"""***# define the optimizer, early stopping and model saving and compile the model  
#adam = optimizers.Adam(lr=lr, beta\_1=beta\_1, beta\_2=beta\_2, epsilon=epsilon)  
  
#1)compile the model*opt = SGD(lr)  
model.compile(loss = **"binary\_crossentropy"**, optimizer = opt,metrics=[**'accuracy'**])  
  
*#model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# add early stopping*monitor = EarlyStopping(monitor=**'val\_loss'**, min\_delta=1e-4, patience=5, verbose=1, mode=**'auto'**)  
*#A checkpointer is a mechanic to save the trained parameters*checkpointer = ModelCheckpoint(**'best.h5'**,  
 monitor=**'val\_loss'**, save\_best\_only=True, verbose=1)  
*# Train the model*history = model.fit(x\_train, y\_train, validation\_data=(x\_val,y\_val), epochs=epochs, batch\_size=bs, callbacks=[monitor, checkpointer])  
  
*#load the saved best weights for further analysis*model.load\_weights(**"best.h5"**)  
  
**"""## 6. Visualize"""***# plot train and validation loss*plt.plot(history.history[**'loss'**])  
plt.plot(history.history[**'val\_loss'**])  
plt.title(**'model loss'**)  
plt.ylabel(**'loss'**)  
plt.xlabel(**'epoch'**)  
plt.legend([**'train'**, **'validation'**], loc=**'upper left'**)  
plt.show()  
plt.close()  
  
*######################################################prediction#########################################################*y\_pred = model.predict(x\_test)  
test\_loss, test\_acc = model.evaluate(x\_test, y\_test)*#test loss-less critical-need to look about test accurac->1  
# Print results***print**(**'test loss:'**, test\_loss)  
**print**(**'test accuracy:'**, test\_acc)  
  
  
  
*# #plot confusion matrix*cm=confusion\_matrix(y\_test.argmax(axis=1),np.round(y\_pred.argmax(axis=1)))  
  
plt.clf()  
plt.imshow(cm, interpolation=**'nearest'**, cmap=plt.cm.Wistia)  
classNames = [**'Negative'**,**'Positive'**]  
plt.title(**'Versicolor or Not Versicolor Confusion Matrix - Test Data'**)  
plt.ylabel(**'True label'**)  
plt.xlabel(**'Predicted label'**)  
tick\_marks = np.arange(len(classNames))  
plt.xticks(tick\_marks, classNames, rotation=45)  
plt.yticks(tick\_marks, classNames)  
s = [[**'TN'**,**'FP'**], [**'FN'**, **'TP'**]]  
**for** i **in** range(num\_of\_clss):  
 **for** j **in** range(num\_of\_clss):  
 plt.text(j,i, str(s[i][j])+**" = "**+str(cm[i][j]))  
plt.show()