Breast Cancer Diagnosis Using Deep Learning Algorithm

Researcher: Maor Karakukli

Institute: Holon Institute Of Technology

Department: Electronics & Electricity Engineering

ID:203281803

1. Introduction:

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)

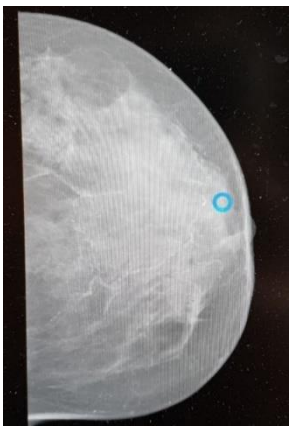
-implement by dicom fies,(***.dcm) .

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!!!**

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

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

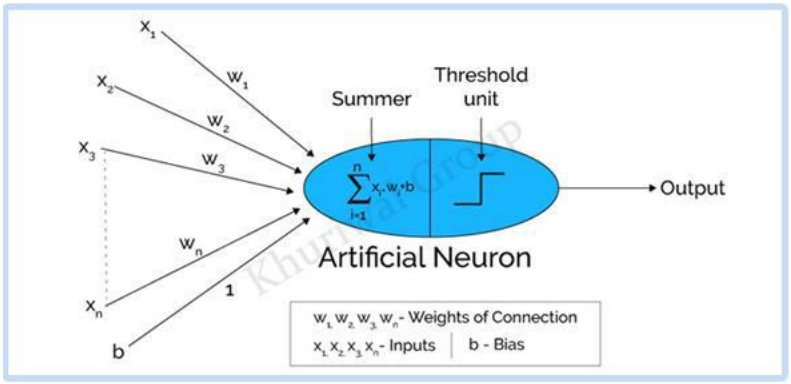
Learning rate:0.01

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.




```
Model: "model_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        (None, 664, 512, 1)       0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 664, 512, 64)      1664
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 332, 256, 64)      0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 332, 256, 32)      51232
_________________________________________________________________
batch_normalization_28 (Batc (None, 332, 256, 32)      128
_________________________________________________________________
dropout_28 (Dropout)         (None, 332, 256, 32)      0
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 166, 128, 32)      0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 166, 128, 32)      25632
_________________________________________________________________
batch_normalization_29 (Batc (None, 166, 128, 32)      128
_________________________________________________________________
dropout_29 (Dropout)         (None, 166, 128, 32)      0
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 83, 64, 32)        0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 83, 64, 32)        25632
_________________________________________________________________
batch_normalization_30 (Batc (None, 83, 64, 32)        128
_________________________________________________________________
dropout_30 (Dropout)         (None, 83, 64, 32)        0
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 41, 32, 32)        0
_________________________________________________________________
flatten_10 (Flatten)         (None, 41984)             0
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 83970
=================================================================
Total params: 188,514
Trainable params: 188,322
Non-trainable params: 192
_________________________________________________________________
```

```
# define a convolutional network
inp = Input(shape = [x_train.shape[1],x_train.shape[2], x_train.shape[3]])
pic_conv_kernel_size = 5

pic_1 = Conv2D(64, pic_conv_kernel_size, padding='same', strides=1, activation='relu')(inp)

pic_2 = MaxPooling2D(pool_size=2)(pic_1)


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)


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)

pic_6 = MaxPooling2D(pool_size=2)(x)


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)

pic_8 = MaxPooling2D(pool_size=2)(x4)

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()
```

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,

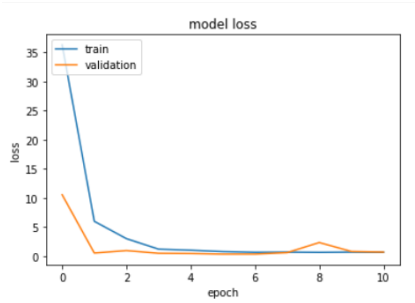
And using dense and flatten in the model,and also "softmax" as Activation(F.C layer),

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

Summary of the Model:

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

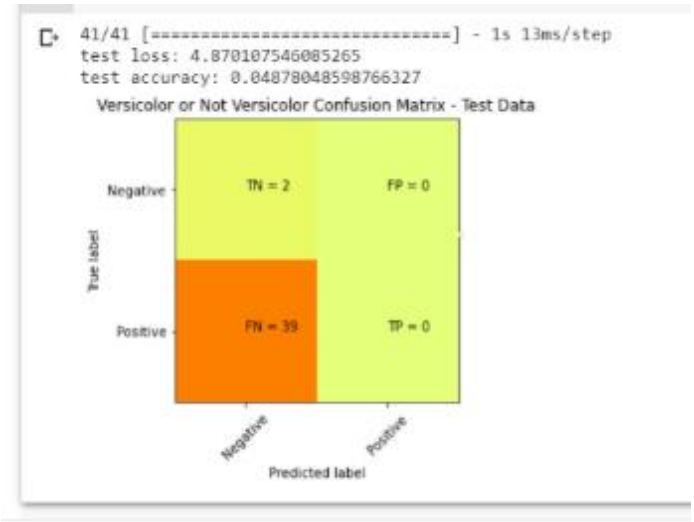
Training results



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):

```
500/500 [==============================] - 0s 132us/step
test loss: 0.5422709391117096
test accuracy: 0.8299999833106995
Normalized confusion matrix
```

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

```
10000/10000 [==============================] - 8s 786us/step
500/500 [==============================] - 0s 132us/step
test loss: 0.5422709391117096
test accuracy: 0.8299999833106995
Normalized confusion matrix
```

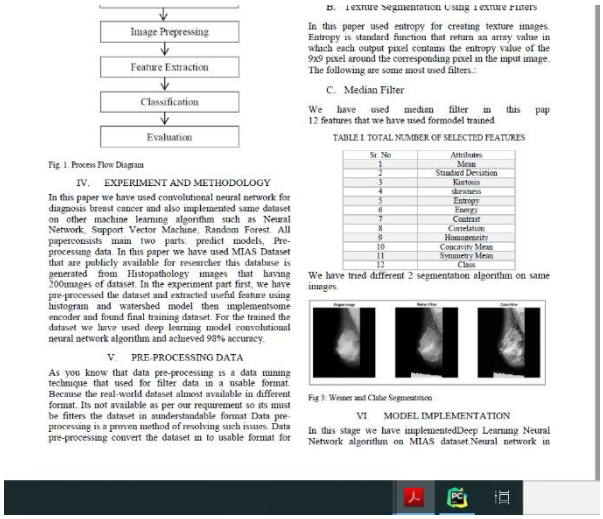
Important: There are methods to overcome that issues, For example: augmentation, Fine-Tuning and etc.

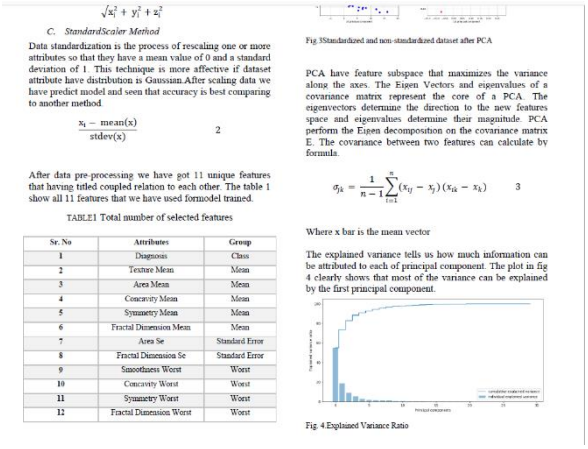
Although I didn't use them according to Time constraints.

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()
```