Islamic University of Technology
Dept. of Computer Science & Engineering

CSE 4710 – Machine Learning Lab

Project Report on Glioblastoma Detection using Convolution Neural Network

Gazi Wasif Akram – 160041005
Sidratul Tamzida Tuba – 160041015
S. M. Rayeed – 160041045

September 18, 2020

Glioblastoma (Brain Tumor) Detection from MRI Images Using Convolution Neural Network

Gazi Wasif Akram
ID : 160041005
wasifakram@iut-dhaka.edu

Sidratul Tamzida Tuba
ID : 160041015
sidratultamzida@iut-dhaka.edu

S. M. Rayeed
ID : 160041045
rayeed@iut-dhaka.edu

***Abstract* - This is a progress report of our Machine Learning project, which is an implementation of a classifier to detect Glioblastoma (Brain Tumor) from MRI images. To do this implementation, we have used convolutional neural network & constructed a CNN model that yields moderate accuracy.**

Keywords – Machine Learning, Neural Networks, Convolutional Neural Networks, Activation Map, Filter, Max-Pooling, Dropout, Feature Engineering, Hyperparameters, Activation Function, Glioblastoma, Image Thresholding, F-measure, Confusion Matrix.

I. INTRODUCTION

This progress report contains the motivation, background study & steps of implementation of our project. In this project, we have tried to incorporate our knowledge of machine learning to that of respective research domains.

A. *Problem Statement*

As mentioned, we have built a model of Convolutional Neural Network for Glioblastoma detection. Glioblastoma is the most aggressive type of brain cancer, containing one of the most invasive type of glial tumors, rapidly growing and commonly spreading into nearby brain tissue. Our model has been trained using a dataset of MRI images, as MRI is one of the ways (along with CT scan and biopsy) to diagnose this fatal disease.

B. *Application Domain*

The problem of our project falls into the Bioinformatics domain, as it is the detection of Tumor cells in Brain. And the implementation approach falls into Deep Learning, as we have implemented this using Neural Network, the core of Deep Learning. So, here applying the knowledge of Deep Learning to Medical Diagnosis is all we have done.

C. *Motivation*

We tried to implement something that is somehow related to our thesis. One of us is doing research on Bioinformatics & is working on glioblastoma analysis. And one is working on Sign Language Recognition under HCI domain. We chose this one because this is related to both of our research domain. This is directly related to Glioblastoma analysis, & for language recognition, one of many approaches is using CNN for classification.

II. BACKGROUND STUDY

A. *Overview of the project*

The project is based on the detection of tumor cells in brain. So, this is a classification problem, having two classes – yes (tumor cells are present) or no. To classify we need a dataset. And, in real life, one of the ways of detecting the tumors is MRI. So, we will be using an image dataset that contains MRI images of the brain. And, as the classifier we will be using Convolutional Neural Network which fits the image dataset best.

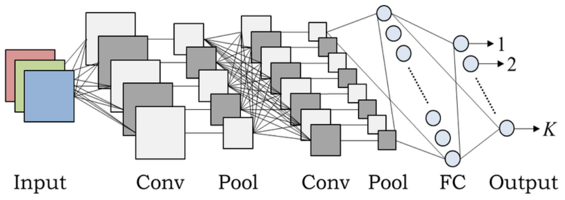
B. *Data collection and feature analysis*

The dataset used in this project is a collection of MRI Images. This is collected from Kaggle, a well-known resource of datasets. The images have been categorized into two categories based on whether there's a tumor cell or not, and have been stored into two folders accordingly. It has a total number of 2264 images, and 1239 of those are classified as Yes, rest as no.

C. *Description of the classification model*

- Classification model – As we've mentioned earlier, we have used Convolutional Neural Network to classify the images. In case of image datasets, CNN outperforms other classifiers. Because architecturally CNN is designed in such a way that it can handle large datasets of images, having images of higher sizes (i.e. higher pixels) by detecting important features & opting out the rest.
- Architectural Diagram – When a dataset of images is fitted to CNN, it considers each pixel as a different feature. The analogy of CNN is, it keeps the important features & opts the others out. To do so, it uses convolution – that's where the name comes from. Convolution is done using kernels, or filters which filtrates the pixels that are important for feature detection & removes the rest. Also, pooling is combinedly used with convolution most often – pooling

further reduces the image size by keeping only the most significant pixel of a pixel-block. These combinedly are called Feature Engineering of CNN. Finally comes the flatten, full connection & output – which just the replicates the usual artificial neural network. Architectural Diagram of CNN given below –



- Parameters & Hyperparameters – in CNN, weights & bias are the parameters, that the machine learns & updates after each epoch. The hyperparameters are number of convolution layers, number of pooling layers, dropout, number of nodes in output layer, activation function in output layer, number of filters in a convolution layer, kernel size, stride, image shape, padding, activation function etc. Hyperparameters need to be tuned in such a way that the model fits well on the dataset, reduce overfitting or underfitting & eventually yields a good result.

III. IMPLEMENTATION

A. *Overview of the Experiment*

Prior to train the model, we need to preprocess the data so that the features can be detected easily from the image samples. After that the dataset needs to be loaded & split into training set & testing set. Then according to the architecture, we need to construct the model & fit the training samples to train the model. Then the hyperparameters need to be tuned. Finally, we'll have to evaluate the performance of the model in testing samples & generate the result.

B. *Data Preprocessing*

As MRI Images are black & white, the machine reads an image as an array of pixels where each pixel has only one channel (colored image has 3 – red, green, blue) & value between 0 to 255, 0 being black & 255 being white. As a part of preprocessing first of all, we have done the image thresholding, and we used binary thresholding. Here, any pixel having a value higher than the threshold will be considered as white & otherwise black. This helps to detect the tumors from the image more clearly.

Secondly, we have used dilation to dilate the noisy parts of the images. Then we have used eroding to erode the foreground boundary. Finally, we have resized the images to a standard size of 256 by 256.

C. *Loading data and Train & Test set generation*

After preprocessing, the samples in the dataset are loaded and then split into Training & Testing set. Splitting ratio is important, because the model needs to be trained with sufficient training samples to yield a good accuracy in case of testing data. We have taken train-test ratio as 8:2, a standard ratio.

D. *Tuning the Hyperparameters*

- Number of Convolution Layers – Too many layers will cause overfitting & computation time will be high. Too less layers will cause underfitting. We have selected two convolution layers.
- Pooling Layers – There're several kinds of pools – such as MaxPool, MinPool, AvgPool, SumPool etc. Among them, MaxPool represents the pooling block most well as we take the pixel with maximum value in that block. Pooling size is also important, because taking size too small will take high computation time & too big will cause losing lots of information as we'll be ignoring lots of pixels in that case. So, we have taken a standard size of 2 by 2.
- Activation function – in convolution layers, we have selected the ReLu function which is usually used. ReLu layer prevents the linear progression across the pixels & introduces non-linearity.
- Input Image Size – Taking higher value might cause high computation time & overfitting whereas too smaller value might cause underfitting. We've taken 256 by 256 as input image shape.
- Number of Filters – We've taken 32 Filters. Taking too many filters might cause overfitting & too less filters might cause excess loss of information.
- Kernel Size – We've taken a standard size of 3 by 3.
- Dropout – Dropout is important because nodes with very smaller weight may cause Vanishing Gradient Problem. We've taken a dropout rate of 0.25 in convolution layers & 0.2 in full connection layer.
- Nodes in Full Connection Layer – taking too many nodes might cause overfitting & high computation time and too less nodes might cause underfitting. We've taken 32.

- Output Layer Activation function – Since this is a two-class classification problem, we have taken sigmoid function.
- Optimizer – selected 'rmsprop'.
- Loss Function – Since this is a two-class classification problem, we have taken binary cross entropy as loss function.

E. *Running the classifier*

After tuning the hyperparameters, we need to fit the training samples to train the model. Training accuracy gives us an overview about how well the model has generalized the training samples. The results show us that, with higher epochs the training accuracy increases, which means the model is trained gradually over the course of epochs. Also, while training we have taken mini batch gradient descent, because stochastic might do little better but at the expense of a lot more computation. On the other hand, for batch gradient descent needs more epochs as it updates the parameters after one whole epoch. Also, it can suffer from local maxima or minima. Considering all, we have selected the batch size to be 32. We have trained the model using different epochs (e.g. 1, 5, 10) & tested the model for testing set for each case. Taking too many would consume a lot of time & the accuracy may increase a little or even may not. So, considering the trade-off, for our final model we have decided to train the model for 10 epochs.

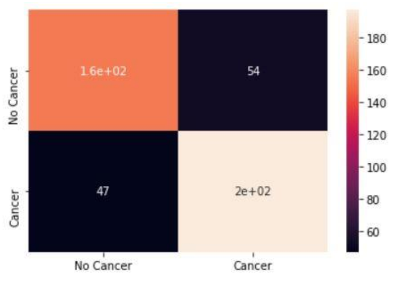
IV. Result Analysis

A. *Overview*

Training the model with 5 epochs gives us an accuracy of about 87% in training dataset & about 76% in testing dataset. Training the same model with 10 epochs yields an accuracy of about 98% in training dataset & about 78% in testing dataset. Also, in case of only 1 epoch, the model gives an accuracy of about 65% in training dataset & about 70% in testing dataset.

B. *Confusion Matrix*

Confusion Matrix is a well-known technique for summarizing the performance of a classifier. Classification accuracy alone can be misleading if there's an imbalance in number of observations in each class. Each row of the matrix represents the instances in a predicted class while each column represents in an actual class.



In this project, the total number of testing samples are 453, among which 352 have correctly been classified & rest 101 have been classified wrongly – which yields an accuracy of about 78%. Also, for 10 epochs, the F-measure or, F1 score we got is 0.80, which is quite good.

C. *Analysis*

From the result we have got, we can see that, the more the epochs are, the more the accuracy increases in the training dataset. Accuracy also increases in the testing dataset, but not at that rate, rather it increases slowly. Also, the model is still having the overfitting problem, though most of it has been mitigated by tuning the hyperparameters. From our analysis, the reason behind it is that the dataset has relatively lesser number of samples than required. Overfitting is a common problem for datasets with limited samples. We have tried to reduce the overfitting as much as possible, presumably with more samples in the dataset & more epochs, higher testing accuracy can be achieved.

Also, we see that, for only one epoch, the model underfits. This is because the model is not trained long enough to fit well in the dataset.

References

[1] Understanding of Convolutional Neural Network (CNN)

[2] Building a Convolutional Neural Network (CNN) in Keras

[3] Glioblastoma Overview

[4] Glioblastoma Multiforme

[5] Brain Tumor Detection using Mask R-CNN

[6] Create your first Image Recognition Classifier using CNN, Keras and Tensorflow backend

[7] Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs)

```
from keras.models import Sequential

from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Dense

from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.utils import shuffle

import cv2
import numpy as np
import matplotlib.pyplot as plt
from os import listdir
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g

Enter your authorization code:
4/4AFiKbmZ_lFBZu8q5oYctI7NtrMscTncZBp_tEH2xaPVGpK_OEDYNHc
Mounted at /content/drive
```

```
!unzip -uq "/content/drive/My Drive/ML Project/Brain Tumor Detection/Real Dataset" -d "/content/drive/My Drive/ML Proje
```

```
def data_summary(yes_path, no_path):

    positiveSamples = len(listdir(yes_path))
    negativeSamples = len(listdir(no_path))
    samples = positiveSamples + negativeSamples

    positiveSamplesPercentage = (positiveSamples * 100.0) / samples
    negativeSamplesPercentage = (negativeSamples * 100.0) / samples

    print('Data Summary \n')
    print('Number of Images : ', samples)
    print('Number of Images with Label - yes : ', positiveSamples)
    print('Number of Images with Label - no : ', negativeSamples)
    print('Percentage of Images with Label - yes : %.2f' % positiveSamplesPercentage, '%')
    print('Percentage of Images with Label - no : %.2f' % negativeSamplesPercentage,'%')
```

```
main_path = "/content/drive/My Drive/ML Project/Brain Tumor Detection/Real Dataset"
yes_path = main_path + '/yes'
no_path = main_path + '/no'
```

```
data_summary(yes_path, no_path)
```

```
Data Summary

Number of Images :  2264
Number of Images with Label - yes :  1239
Number of Images with Label - no :  1025
Percentage of Images with Label - yes : 54.73 %
Percentage of Images with Label - no : 45.27 %
```

Image Thresholding

```
def img_thresholding(image) :

    # Convert the image to grayscale, and blur it slightly
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

```

```python
    thresh = cv2.threshold(gray, 45, 255, cv2.THRESH_BINARY)[1]

    #erodes the foreground boundary
    thresh = cv2.erode(thresh, None, iterations=2)

    #remove small regions of noises
    thresh = cv2.dilate(thresh, None, iterations=2)

    new_image = thresh

    return new_image
```

Loading

```python
def load_data(dir_list, image_size):

    # load all images in a directory
    images = []
    labels = []
    image_width, image_height = image_size

    for directory in dir_list:
        for filename in listdir(directory):

            image = cv2.imread(directory + '/' + filename)

            image = img_thresholding(image)

            #cv2.INTER_CUBIC - a bicubic interpolation over 4×4 pixel neighborhood
            image = cv2.resize(image, dsize=(image_width, image_height), interpolation=cv2.INTER_CUBIC)

            image = image/255

            images.append(image)

            if directory[-3:] == 'yes':
                labels.append([1])
            else:
                labels.append([0])

    images = np.array(images)
    labels = np.array(labels)

    images, labels = shuffle(images, labels)

    return images, labels
```

```python
width, height = (256, 256)

images, labels = load_data([yes_path, no_path], (width, height))
```

```python
from tensorflow.keras import backend

if backend.image_data_format() == 'channels_first':
    images = images.reshape(images.shape[0], 1, 256, 256)
    input_shape = (1, 256, 256)

else:
    images = images.reshape(images.shape[0], 256, 256, 1)
    input_shape = (256, 256, 1)

images = images.astype('float32')
```

```python
train_images, test_images, train_labels, test_labels = train_test_split(images, labels, test_size = 0.2)
```

```python
print('Number of Training Samples : ', train_images.shape[0])
print('Number of Testing Samples : ', test_images.shape[0])
```

```
Number of Training Samples :  1811
Number of Testing Samples :  453
```

```python
model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))
```

```python
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
```

```python
model.fit(train_images, train_labels, batch_size=32, epochs=10)
```

```
Epoch 1/10
57/57 [==============================] - 104s 2s/step - loss: 0.9965 - accuracy: 0.6499
Epoch 2/10
57/57 [==============================] - 104s 2s/step - loss: 0.5171 - accuracy: 0.7620
Epoch 3/10
57/57 [==============================] - 104s 2s/step - loss: 0.4487 - accuracy: 0.7918
Epoch 4/10
57/57 [==============================] - 103s 2s/step - loss: 0.3647 - accuracy: 0.8404
Epoch 5/10
57/57 [==============================] - 103s 2s/step - loss: 0.2824 - accuracy: 0.8807
Epoch 6/10
57/57 [==============================] - 104s 2s/step - loss: 0.1880 - accuracy: 0.9260
Epoch 7/10
57/57 [==============================] - 103s 2s/step - loss: 0.1177 - accuracy: 0.9575
Epoch 8/10
57/57 [==============================] - 105s 2s/step - loss: 0.0891 - accuracy: 0.9663
Epoch 9/10
57/57 [==============================] - 105s 2s/step - loss: 0.0514 - accuracy: 0.9840
Epoch 10/10
57/57 [==============================] - 104s 2s/step - loss: 0.0334 - accuracy: 0.9884
<tensorflow.python.keras.callbacks.History at 0x7f0b95efc710>
```

```python
results = model.evaluate(test_images, test_labels, batch_size=1)

testLoss = results[0]
testAccuracy = results[1]*100

print('Test loss : %.2f' % testLoss)
print('Test accuracy : %.2f' % testAccuracy, "%")
```

```
453/453 [==============================] - 9s 20ms/step - loss: 1.0767 - accuracy: 0.7770
Test loss : 1.08
Test accuracy : 77.70 %
```

```python
predicted_labels = model.predict(test_images)
predicted_labels = (predicted_labels > 0.5)

fmeasureScore = f1_score(test_labels, predicted_labels)
print('F-measure Score : %.2f' % fmeasureScore)
```

```
F-measure Score : 0.80
```

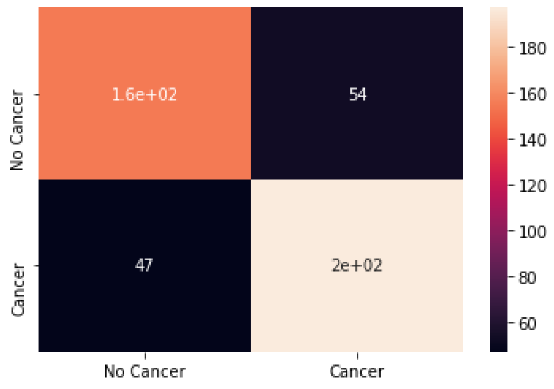
```python
import seaborn as ss
from sklearn.metrics import confusion_matrix

xAxisLabels = ['No Cancer', 'Cancer']
```

```
yAxisLabels = ['No Cancer', 'Cancer']

confusionMatrix = confusion_matrix(test_labels, predicted_labels)
ss.heatmap(confusionMatrix, annot = True, xticklabels=xAxisLabels, yticklabels=yAxisLabels)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b909312b0>
```



```
identificationCorrect = confusionMatrix[0,0] + confusionMatrix[1,1]
identificationIncorrect = confusionMatrix[0,1] + confusionMatrix[1,0]
Identification = identificationCorrect + identificationIncorrect

print('Correct : ', identificationCorrect)
print('Incorrect : ', identificationIncorrect)
```

```
Correct :  352
Incorrect :  101
```