Paper ML13

Application of Machine Learning (ML) in Tumor Detection in Oncology Study

Animesh Patel, NJ, USA
Kamlesh Patel, Rang Technologies LLC, NJ, USA

ABSTRACT

Machine Learning (ML) has been a buzz word nowadays due to its power to bring huge change. However, the application of ML has been minimal in oncology clinical research and healthcare industry. Deep learning is the current trend of the machine learning field. It is widely used in numerous applications for solving complex problems that required more accurate result and sensitivity like in medical field. In oncology studies, the detection of brain tumor (lesion) is performed by investigator and independent reviewer. This method is impractical in large amount of data so tumor detection using Machine learning would save radiologist time. Our work involves develop and implementation of convolution neural network (CNN) for Magnetic resonance imaging(MRI). The dataset we used in this paper have around 3000 images belong four classes where three of different type of tumors and no tumor class. The proposed CNN model gives 97.5% training accuracy and 91.6% testing accuracy. Based on the result our model can be help full to verify patient has tumor or not.

Keywords: Machine Learning (ML), Python, Oncology, Brain Tumor, MRI, CNN, Deep learning, Accuracy.

1. INTRODUCTION

According to WHO Cancer is the second leading cause of death globally and is responsible for about 10 million deaths per year [1]. Brain tumor is defined as collection of cells that grow in an abnormal way. Brain tumor is affecting the central nervous system. Brain tumors have more than 120 different types, according to the National Brain Tumor Society some brain tumor is fast-growing, and some are slow growing. Tumor can be benign or malignant. Malignant tumors lead to cancer while benign tumors are not cancerous. In this paper we used three types brain tumor patient's data. First one is glioma, second is meningioma and pituitary. All different type of tumors has different features and different treatments. Glioma tumor is growing on the area of glia tissues and spinal cord, pituitary tumor grows on pituitary gland area, while meningioma is growing on the area that protects brain and spinal cord 2-3]. The most common and popular method to identify tumor type in magnetic resonance imaging (MRI) is human inspection. Sometime independent assessor and adjudication committee is present. Large amount of data is difficult for human to handle that is the motivation for us to develop CNN model that help and save times. Along with one radiologist, this model can serve the purpose of confirming results i.e. can full fill the need of 2nd radiologist or can serve the purpose of adjudication.

2. METHDOLOGY

2.1 Database

There are many types of tumor are present now a days but in our analysis Brain tumor dataset contain three types of brain tumor patients MRI images, which are Glioma Tumor, Meningioma Tumor and Pituitary Tumor. Along with these three types we also took normal patients MRI images which means patients who hasn't brain tumor. Around 3000 MRI images we used to develop our CNN model. Data is publicly available at on online[4][5].

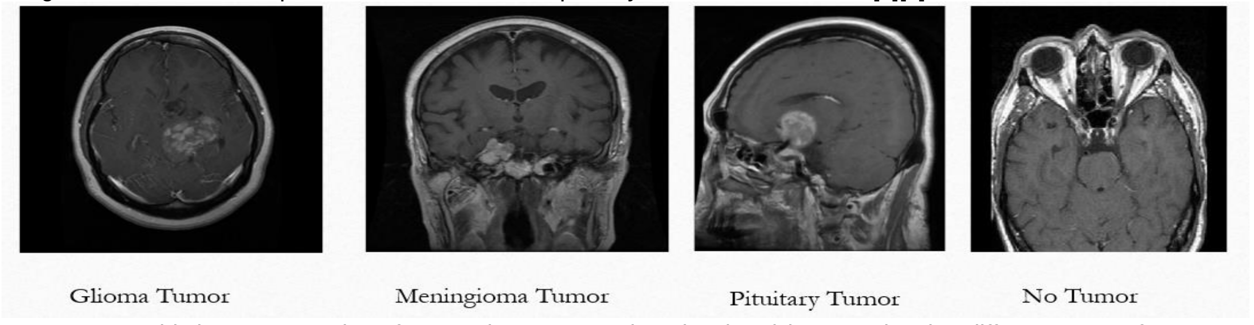


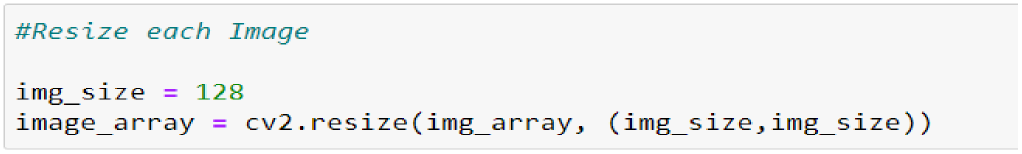
FIG: -1 This is representation of magnetic resonance imaging (MRI) images showing different types of tumors.

2.2 Image Pre-Processing

Whenever we used Images as our data, all the images are not in same size (height, weight), same format so based on this we need to perform some pre-processing step in order to extract the most useful information from images. we performed couple of preprocessing steps like Resize all images, Convert into gray scale and Normalized pixel value.

2.2.1 Resize Images

Convolutional Neural Network has many layers which depends on input size therefor resizing is important. If we give different size of images to CNN model it takes longer time to process all images that's why we give same size of all images which is 128 X 128. Python has Package called OpenCV which provides cv2.resize() function with the help we resize all images as see on the below fig.

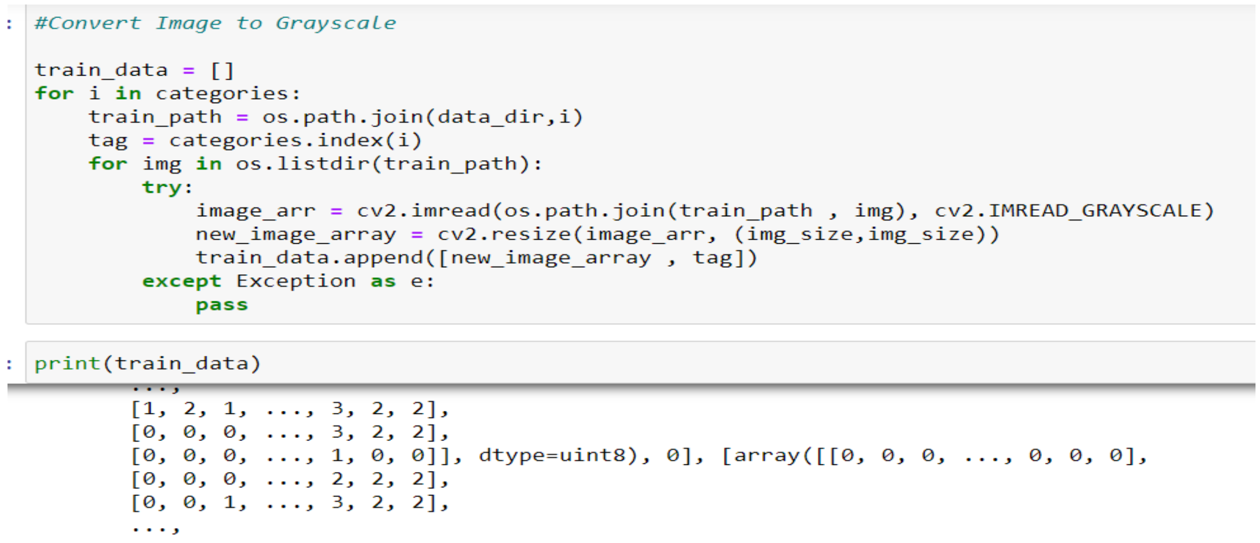


```
#Resize each Image

img_size = 128
image_array = cv2.resize(img_array, (img_size,img_size))
```

FIG: -2 Resize all images to same size.

2.2.2 Convert into Gray Scale

Now we convert the images into Gray scale. Why we need to do this pre-processing step is because in MRI Images unnecessary information are present which we called Noise information or noise data. This information we don't need to pass in our model so after this step only those information extract from images which are important and store in an array as we saw in below picture.



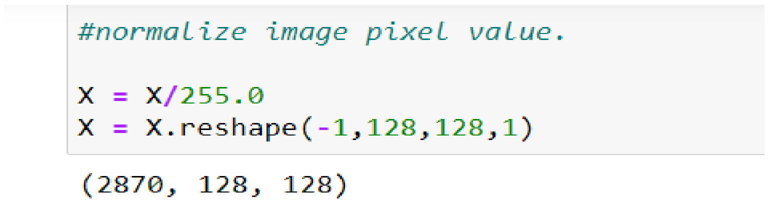
```
: #Convert Image to Grayscale

  train_data = []
  for i in categories:
      train_path = os.path.join(data_dir,i)
      tag = categories.index(i)
      for img in os.listdir(train_path):
          try:
              image_arr = cv2.imread(os.path.join(train_path , img), cv2.IMREAD_GRAYSCALE)
              new_image_array = cv2.resize(image_arr, (img_size,img_size))
              train_data.append([new_image_array , tag])
          except Exception as e:
              pass

: print(train_data)
```

```
        ...,
        [1, 2, 1, ..., 3, 2, 2],
        [0, 0, 0, ..., 3, 2, 2],
        [0, 0, 0, ..., 1, 0, 0]], dtype=uint8), 0], [array([[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 2, 2, 2],
        [0, 0, 1, ..., 3, 2, 2],
        ...,
```

FIG: -3 Convert into Gray scale and store in an array.

2.2.3 Normalized Pixel Value

All the Images have a pixel in range of 0-255. This is not ideal to take large amount of pixel value in neural network. so, hare we need to normalize the pixel value in between 0-1. For that we simply divide the pixel value with largest value(255) as see in below image and we get normalized value in range of 0-1.



```
#normalize image pixel value.

X = X/255.0
X = X.reshape(-1,128,128,1)
```

```
(2870, 128, 128)
```

FIG: -4 Normalized Pixel value

2.2.4 Splitting Data

We split data into training and testing dataset. We used 80% of data for train our model where 20% of data used to test our model. 80-20 are not the fixed number for splitting we can either used 70-30,60-40,85-15 etc. But logic behind this was if we give mode more number of data to train our model, model will train and performed very well while if we give less number in training part model will not learn as much as we think because of less information.

3. APPROACH

We used Python programming with using Keras and TensorFlow packages. These two packages are very important for any of the machine learning and deep learning application. Implement our model on Jupyter notebook. By using convolutional neural network, we performed MRI-image detection.

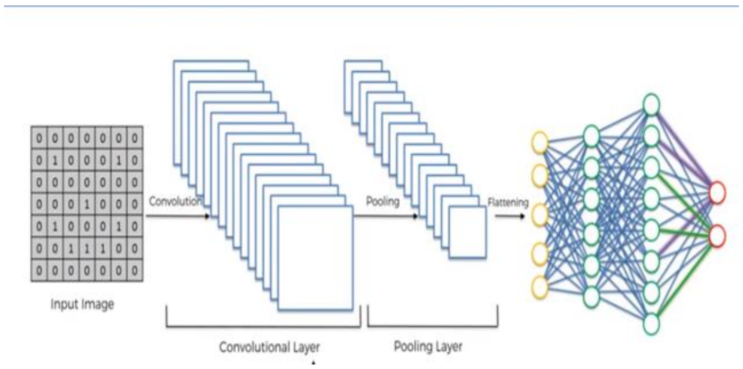
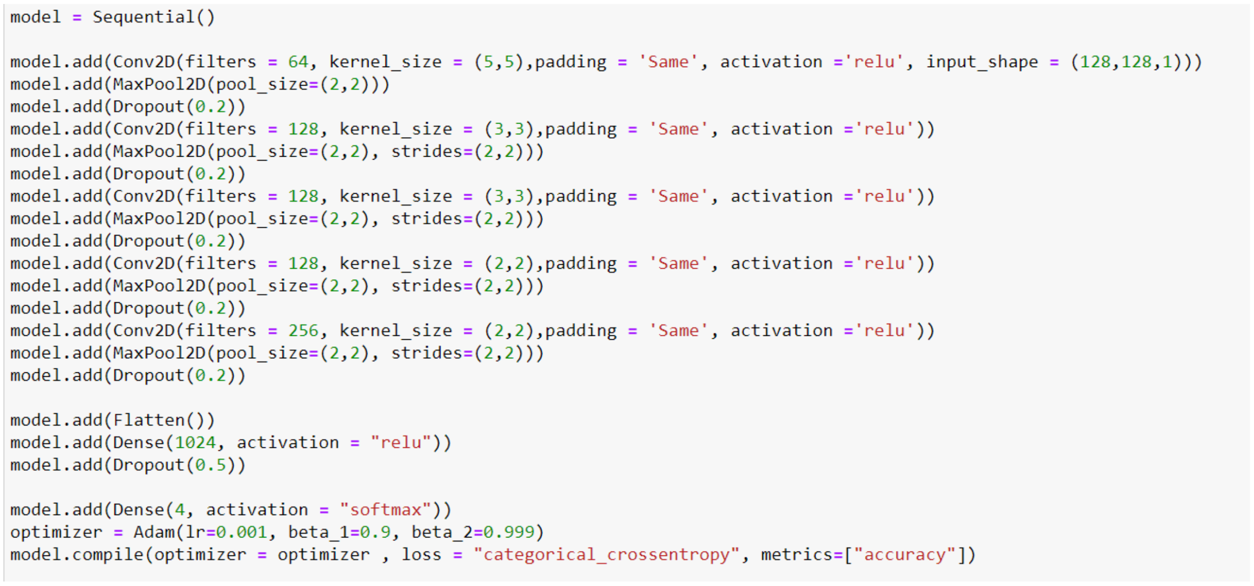


FIG: -5 CNN Architecture [6]

There are many other models and networks are available like Recurrent neural network(RNN), Artificial neural network(ANN) etc. but we used CNN in our analysis because convolutional neural network has the ability to performed image recognition, image classification, object detection, face recognition etc. Above attached image is Simplest architecture of CNN. Where we see there are many layers are involving in CNN. As compared to our scenario input image is MRI image passed through all the layers and at the end it will detect patient has Tumor or not.

Four important layers in CNN are Convolution layer, ReLU layer, Pooling layer, Fully connected layer. All this layer has own different task to performed. Convolutional layer is core block of CNN. Objective of this layer is extract features from input image and store in array which known as feature map. Second layer is relu layer it Is applying onto feature map to increased non-linearity in the network. It also helps to remove negative value from feature map by setting them to zero. Objective of pooling layer is reducing the spatial size and number of parameters. Pooling has two methods max pooling and average pooling. Pooling also help to control overfitting. Last layer is fully connected layer it identifies the final output, patient has tumor or not based on previous layers parameters.



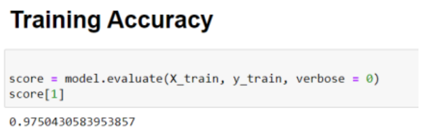
```
model = Sequential()

model.add(Conv2D(filters = 64, kernel_size = (5,5),padding = 'Same', activation ='relu', input_shape = (128,128,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same', activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 128, kernel_size = (3,3),padding = 'Same', activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 128, kernel_size = (2,2),padding = 'Same', activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(filters = 256, kernel_size = (2,2),padding = 'Same', activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(1024, activation = "relu"))
model.add(Dropout(0.5))

model.add(Dense(4, activation = "softmax"))
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
model.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
```

FIG: -6 Created CNN Model

Fig-6 is our created CNN model. In our model architecture involves five convolutional layers followed by max-pooling layer and dropout layer. No of filter in each block is increased with the depth of network such as 64,128,256. Padding is used on convolutional layer to ensure that height and width of output feature map is same as input feature map. Relu activation we used. Dropout is a technique that will randomly drop nodes from the network. Dropout 0.2 means 20% of unnecessary data model will remove. Last, we add fully connect dense layer with softmax activation function. Model will be optimized using Adam optimizer.

4. RESULT

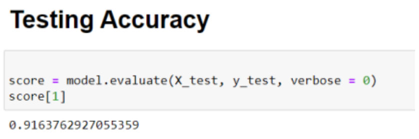
After creating CNN model, we run our model and find out the best result. We run all the MRI images 20 times means 20 epochs to get highest training and testing accuracy. 97.5% and 91.6% are training and testing accuracy we obtained. We also investigate training and testing Loss to check weather our model goes to underfit or overfit.



Training Accuracy

```
score = model.evaluate(X_train, y_train, verbose = 0)
score[1]
```

```
0.9750430583953857
```



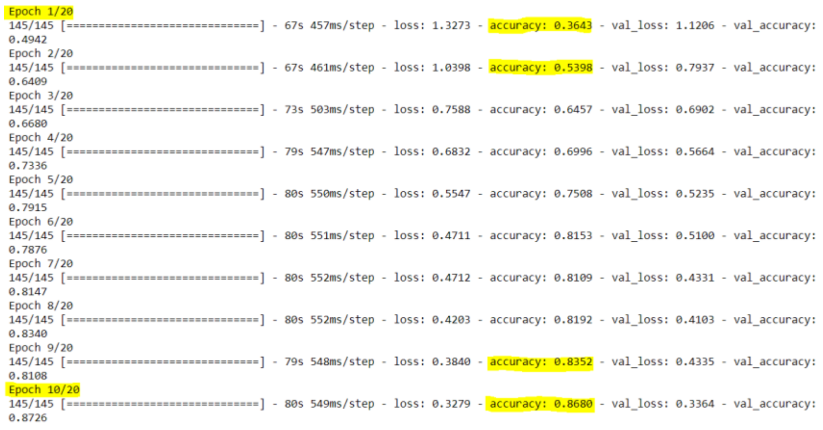
Testing Accuracy

```
score = model.evaluate(X_test, y_test, verbose = 0)
score[1]
```

```
0.9163762927055359
```

FIG: - 7 Accuracy

As per the Fig-8 we can said that after completing first epoch model training accuracy is 36.43% and testing accuracy is 49.42%. as umber of epoch increased, we see accuracy also increase after 10Th epoch training and testing accuracy we get is 86.80% and 87.26%. Training and testing Loss values constantly decrease.



```
Epoch 1/20
145/145 [==============================] - 67s 457ms/step - loss: 1.3273 - accuracy: 0.3643 - val_loss: 1.1206 - val_accuracy:
0.4942
Epoch 2/20
145/145 [==============================] - 67s 461ms/step - loss: 1.0398 - accuracy: 0.5398 - val_loss: 0.7937 - val_accuracy:
0.6409
Epoch 3/20
145/145 [==============================] - 73s 503ms/step - loss: 0.7588 - accuracy: 0.6457 - val_loss: 0.6902 - val_accuracy:
0.6680
Epoch 4/20
145/145 [==============================] - 79s 547ms/step - loss: 0.6832 - accuracy: 0.6996 - val_loss: 0.5664 - val_accuracy:
0.7336
Epoch 5/20
145/145 [==============================] - 80s 550ms/step - loss: 0.5547 - accuracy: 0.7508 - val_loss: 0.5235 - val_accuracy:
0.7915
Epoch 6/20
145/145 [==============================] - 80s 551ms/step - loss: 0.4711 - accuracy: 0.8153 - val_loss: 0.5100 - val_accuracy:
0.7876
Epoch 7/20
145/145 [==============================] - 80s 552ms/step - loss: 0.4712 - accuracy: 0.8109 - val_loss: 0.4331 - val_accuracy:
0.8147
Epoch 8/20
145/145 [==============================] - 80s 552ms/step - loss: 0.4203 - accuracy: 0.8192 - val_loss: 0.4103 - val_accuracy:
0.8340
Epoch 9/20
145/145 [==============================] - 79s 548ms/step - loss: 0.3840 - accuracy: 0.8352 - val_loss: 0.4335 - val_accuracy:
0.8108
Epoch 10/20
145/145 [==============================] - 80s 549ms/step - loss: 0.3279 - accuracy: 0.8680 - val_loss: 0.3364 - val_accuracy:
0.8726
```

FIG: - 8 10-EPOCH Result

Python has matplotlib package with the help of this package we plot these two graphs which are training, and testing Accuracy vs No of Epoch and second graph is training and testing loss vs No of Epoch. After seeing this graph, we conclude that no of epoch are increased training and testing accuracy also increased. On other side training and testing loss are decreased while number of epochs increased. That is the sign of our model give us an optimal result.

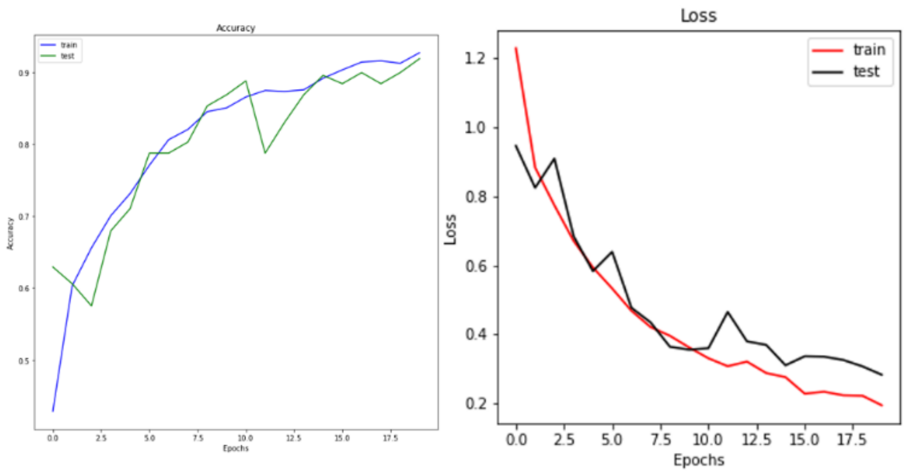


FIG: - 9 Accuracy vs Epoch & Loss vs Epoch

5.CONCLUSION

Detection of brain tumors has a very significant role to play in medical interventions. We are able to create and train model that gave us very high accuracy. This model can be taken and developed to specific need of company. With the using of this automated detection model, it will help to save radiologist time and saving cost in clinical research. Can be good option for adjudication (3rd Radiologist) in oncology clinical trial.

REFERENCES

1. https://www.who.int/news-room/fact-sheets/detail/cancer
2. https://www.cancer.net/cancer-types/brain-tumor/view-all
3. https://www.cancercenter.com/cancer-types/brain-cancer/types
4. https://figshare.com/articles/dataset/brain_tumor_dataset/1512427
5. https://github.com/SartajBhuvaji/Brain-Tumor-Classification-DataSet
6. https://www.ripublication.com/ijaer18/ijaerv13n20_26.pdf
7. https://en.wikipedia.org/wiki/Convolutional_neural_network

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Animesh Patel
Email: animeshdatascience@gmail.com
Company: Independent Consultant
Address: Piscataway, NJ
City / Postcode: 08854