Summer Internship at C-DAC, Pune

Project Title	**Segmentation of brain tumor images using Deep learning methods**
Duration of Internship	1 months
Date of Submission of Project	25th JULY 2018
Name of Institute	Manipal University Jaipur, Jaipur
Submitted by	Lakshita Bhargava
Department	Information Technology
Discipline	B.TECH 3rd Year

PROJECT GUIDE

Mr. Amit Saxena
Senior Technical Officer
HPC: Medical and Bioinformatics Applications
C-DAC, Pune

SUBMITTED BY

Lakshita Bhargava
Manipal University Jaipur,Jaipur

Table of Contents

Objectives of Project

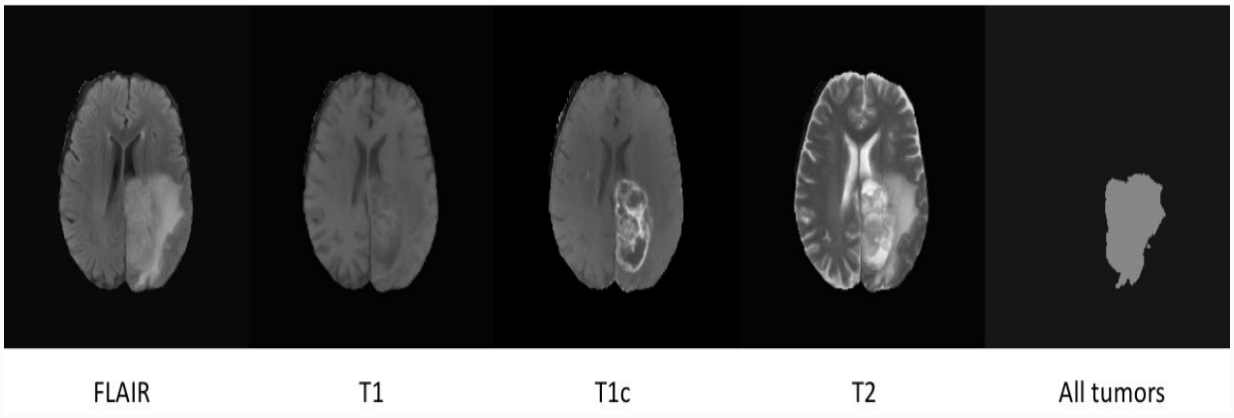
1. To implement CNN on MRI Images

Brain Tumor

Brain tumor detection is an algorithm for identifying the tumor present in the Brain. Brain tumor patients often suffer from blood clot, movement control loss, vision loss, behavioral changes, hormone changes, etc. The location, type and size of the tumor have an effect on the normal functioning of the individual. MRI images help the doctors for identifying the Brain tumor size and shape of the tumor. But, it consumes the doctor's time. In order to save the time and burden of the doctor, there is a need for the automation of the brain tumor.

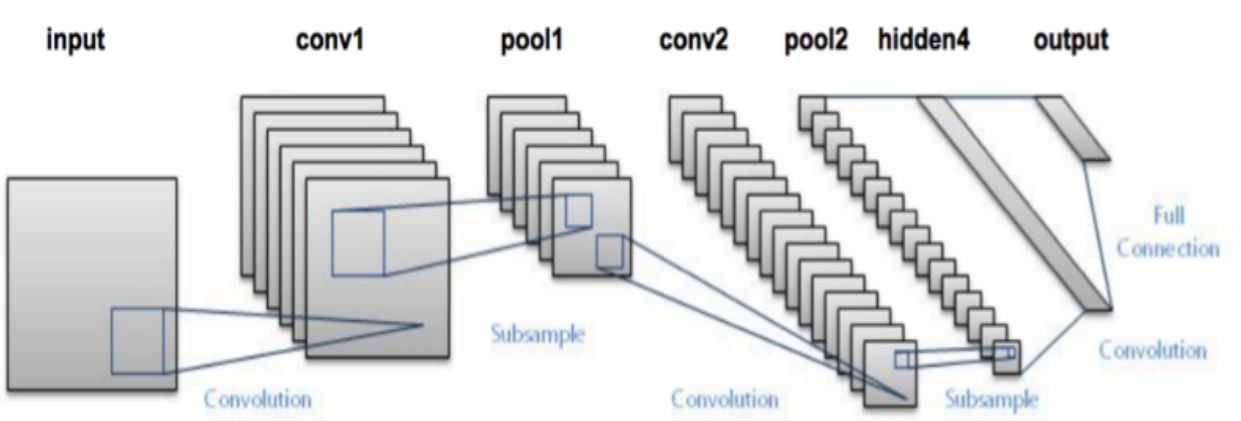
Dataset

- Multimodal Brain Tumor Image Segmentation Benchmark (BRATS) organized in conjunction with the MICCAI conferences.
- It has both High grade glioma(HGG) and low grade glioma(LGG)
- Images are in .mha format
- .mha format is a simple image format that can be used to handle n-dimensional image data.



What is CNN?

CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function.
CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. They are also known as **shift invariant** or **space invariant artificial neural networks** (**SIANN**), based on their shared-weights architecture and translation invariance characteristics.
CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. Here the input is a multi-channeled image.



Types of Layers

We use three main types of layers to build ConvNet architectures: **Convolutional Layer, Pooling Layer,** and **Fully-Connected Layer.**

1. Convolutional Layer: It forms the basis of the CNN and performs the core operations of training and consequently firing the neurons of the network. It performs the convolution operation over the input volume.

   Conv3D : (Spatial convolutional over Volume) This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

   keras.layers.Conv3D(filters, kernel_size, strides=(1, 1, 1),padding='valid', data_format=**None**, dilation_rate=(1, 1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

   - **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
   - **kernel_size**: An integer or tuple/list of 3 integers, specifying the depth, height and width of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
   - **strides**: An integer or tuple/list of 3 integers, specifying the strides of the convolution along each spatial dimension. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
   - **padding**: one of "valid" or "same" (case-insensitive).
   - **data_format**: A string, one of "channels_last" or "channels_first". The ordering of the dimensions in the inputs."channels_last" corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)while "channels_first" corresponds to inputs with shape(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
   - **dilation_rate**: an integer or tuple/list of 3 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

- **activation**: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: a(x) = x).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix (see initializers).
- **bias_initializer**: Initializer for the bias vector (see initializers).
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see regularizer).
- **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint**: Constraint function applied to the kernel matrix (see constraints).
- **bias_constraint**: Constraint function applied to the bias vector (see constraints).

2. **Pooling Layer** : Max pooling operation for 3D data (spatial or spatio-temporal).

keras.layers.MaxPooling3D(pool_size=(2, 2, 2), strides=**None**, padding='valid', data_format=**None**)

- **pool_size**: tuple of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension.
- **strides**: tuple of 3 integers, or None. Strides values.
- **padding**: One of "valid" or "same" (case-insensitive).
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs.channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels)while channels_first corresponds to inputs with shape(batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last"

3. **Fully-Connected Layer**:
   Dense: Just your regular densely-connected NN layer.
   Dense implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

   Activation: Applies an activation function to an output.
   a. Relu: Rectified Linear Unit.
      Arguments

      - **x**: Input tensor.
      - **alpha**: Slope of the negative part. Defaults to zero.
      - **max_value**: Maximum value for the output.

   b. Softmax: Softmax function calculates the probabilities distribution of the event over 'n' different events
      Arguments
      - **x**: Input tensor.
      - **axis**: Integer, axis along which the softmax normalization is applied.

   DropOut: Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

Why CNN?

The usage of CNNs are motivated by the fact that they can capture / are able to learn relevant features from an image /video at different levels similar to a human brain.The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The neurons inside a layer are connected to only a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.

Local connectivity: following the concept of receptive fields, CNNs exploit spatial locality by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear filters that become increasingly global (i.e. responsive to a larger region of pixel space) so

that the network first creates representations of small parts of the input, then from them assembles representations of larger areas.

Shared weights: In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer respond to the same feature within their specific response field.

Libraries Used:

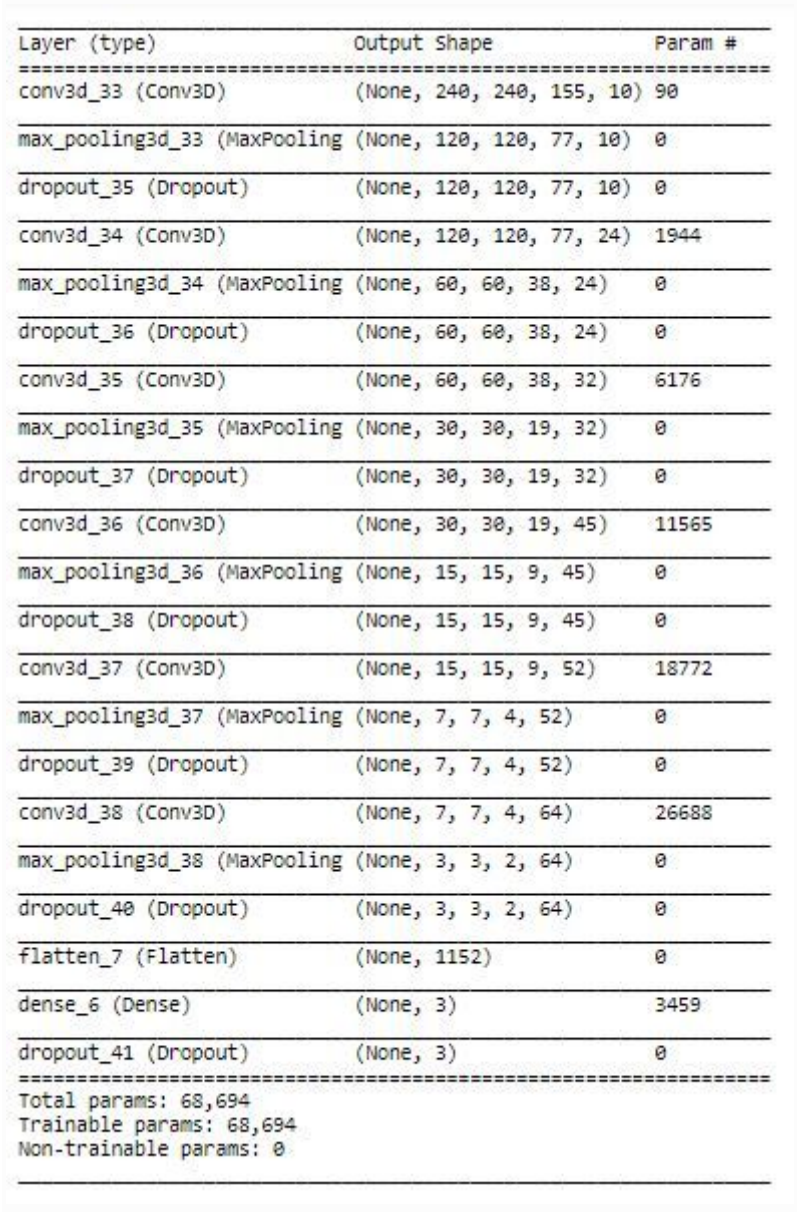
- **KERAS :** Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.
- **TENSORFLOW :** TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.
- **NUMPY :** Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **NIBABEL:** NiBabel is the successor of PyNIfTI. The various image format classes give full or selective access to header (meta) information and access to the image data is made available via NumPy arrays.

Implementation:

```
#import libraries to make model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv3D, MaxPooling3D, GlobalAveragePooling3D

batch_size=14
epochs=10
n=2 #no.of classes

model=Sequential()
model.add(Conv3D(filters=10, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True, input_shape=
          ))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
model.add(Conv3D(filters=24, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
model.add(Conv3D(filters=32, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
model.add(Conv3D(filters=45, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
model.add(Conv3D(filters=52, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
model.add(Conv3D(filters=64, kernel_size=(2,2,2), strides=(1,1,1), padding='same', activation='relu', use_bias=True))
model.add(MaxPooling3D(pool_size=(2,2,2)))
model.add(Dropout(0.25))
#model.add(GlobalAveragePooling3D())
model.add(Flatten())
model.add(Dense(3, activation='relu'))
#model.add(Flatten())
#model.add(activation='softmax')
model.add(Dropout(0.25))

model.summary()
```

6 Convolutional Layers are added with activation as relu and increasing filter size.Max Pooling is also added as It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the **maximum**. Dropout is added to remove any overfitting present in the architecture.
Flatten is used to flatten the output and feed it into Dense (fully connected layer) which uses relu as activation function. At the end a dropout is added to remove any further overfitting.

Output:



Layer (type)	Output Shape	Param #
conv3d_33 (Conv3D)	(None, 240, 240, 155, 10)	90
max_pooling3d_33 (MaxPooling	(None, 120, 120, 77, 10)	0
dropout_35 (Dropout)	(None, 120, 120, 77, 10)	0
conv3d_34 (Conv3D)	(None, 120, 120, 77, 24)	1944
max_pooling3d_34 (MaxPooling	(None, 60, 60, 38, 24)	0
dropout_36 (Dropout)	(None, 60, 60, 38, 24)	0
conv3d_35 (Conv3D)	(None, 60, 60, 38, 32)	6176
max_pooling3d_35 (MaxPooling	(None, 30, 30, 19, 32)	0
dropout_37 (Dropout)	(None, 30, 30, 19, 32)	0
conv3d_36 (Conv3D)	(None, 30, 30, 19, 45)	11565
max_pooling3d_36 (MaxPooling	(None, 15, 15, 9, 45)	0
dropout_38 (Dropout)	(None, 15, 15, 9, 45)	0
conv3d_37 (Conv3D)	(None, 15, 15, 9, 52)	18772
max_pooling3d_37 (MaxPooling	(None, 7, 7, 4, 52)	0
dropout_39 (Dropout)	(None, 7, 7, 4, 52)	0
conv3d_38 (Conv3D)	(None, 7, 7, 4, 64)	26688
max_pooling3d_38 (MaxPooling	(None, 3, 3, 2, 64)	0
dropout_40 (Dropout)	(None, 3, 3, 2, 64)	0
flatten_7 (Flatten)	(None, 1152)	0
dense_6 (Dense)	(None, 3)	3459
dropout_41 (Dropout)	(None, 3)	0

Total params: 68,694
Trainable params: 68,694
Non-trainable params: 0

The above image is the summary of model.

Conclusion:

The model was successfully applied on the images. Apart from brain tumor, ccn2d model for dog prediction was also implemented to get a basic idea of how to make a cnn model.

Future Work:

- More layers can be added in the architecture to make it better.
- Pre trained models like ResNet50, Inception, VGG16 can be used to increase the accuracy

References:

1. https://sites.google.com/site/braintumorsegmentation/
2. https://github.com/Kamnitsask/deepmedic
3. https://www.smir.ch/BRATS/Start2015
4. https://github.com/lelechen63/MRI-tumor-segmentation-Brats
5. https://www.med.upenn.edu/sbia/brats2017.html
6. https://www.nitrc.org/frs/?group_id=546
7. http://brainweb.bic.mni.mcgill.ca/brainweb/
8. https://ieeexplore.ieee.org/document/6975210/
9. http://casemed.case.edu/clerkships/neurology/Web%20Neurorad/MRI%20Basics.htm