ED6001: Medical Image Analysis

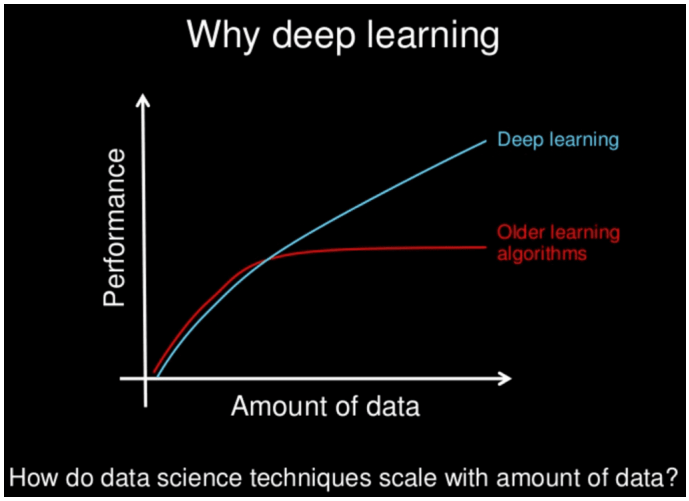
Term Paper Report

Brain Tumour Classification using Deep learning Approach

Submitted by:
Shital Yelne EE20S052
Piyush Nehete CL21M011

Deep Learning

Deep learning can be thought of as a way to automate predictive analytics.It excels in recognizing objects in images as it's implemented using 3 or more layers of artificial neural networks where each layer is responsible for extracting one or more features of the image.
Deep learning has absolutely dominated computer vision over the last few years.
Over the past few years, deep learning techniques have enabled rapid progress even surpassing human performance.
source:https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef



Also there are several other techniques being introduced in deep learning which makes it possible to predict using less data. We tried exploring transfer learning in this project which is one such method.

Problem Statement

We are having brain data containing different types of tumours and normal tumour data. There are four folders of three different types of tumours *i.e no_tumor,glioma_tumor ,meningioma_tumor and pituitary_tumor. Other folder has normal brain data(no_tumour)*. Training and Testing samples have been provided. No ground truth images are given.

So we pose this as a Multiclass Classification Problem which we tried to solve using different deep learning approaches.

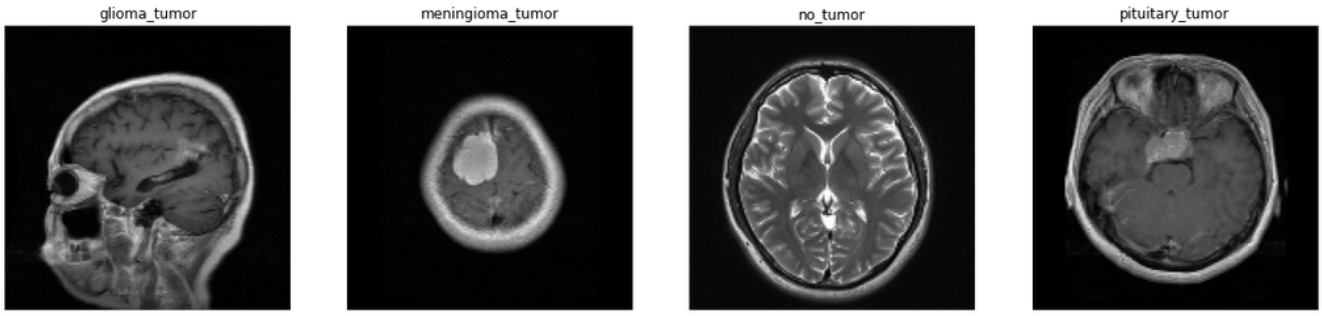
Dataset

Dataset is having four different classes i.e. 'glioma_tumor', 'meningioma_tumor', 'no_tumor', 'pituitary_tumor'

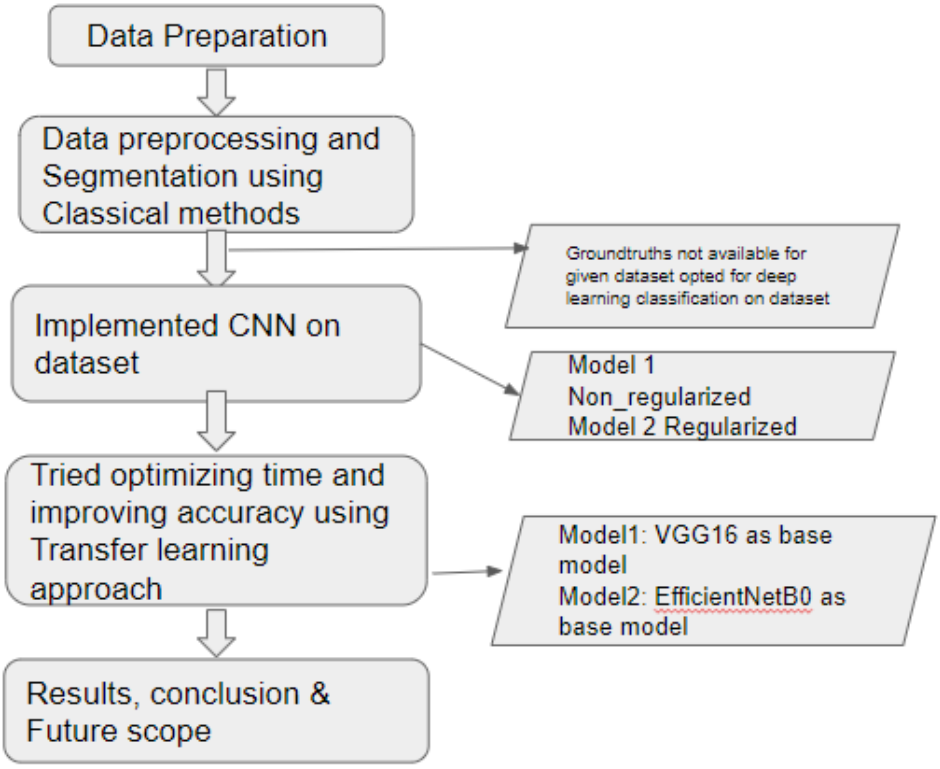
Training :
glioma_tumor= 826 samples
meningioma_tumor= 822 samples
no_tumor= 395 samples
pituitary_tumor= 827 samples

Test:
glioma_tumor= 100 samples
meningioma_tumor= 115 samples
no_tumor= 105 samples
pituitary_tumor= 74 samples

Images one of each type:



Workflow:

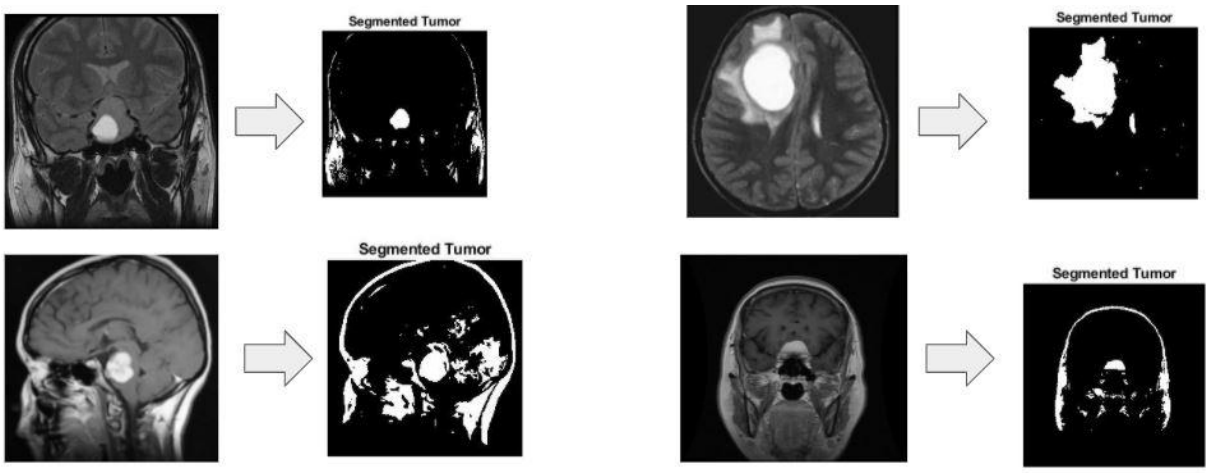

Data Preparation
Data preprocessing and Segmentation using Classical methods
Groundtruths not available for given dataset opted for deep learning classification on dataset
Implemented CNN on dataset
Model 1
Non_regularized
Model 2 Regularized
Tried optimizing time and improving accuracy using Transfer learning approach
Model1: VGG16 as base model
Model2: EfficientNetB0 as base model
Results, conclusion & Future scope

Segmentation

Process of Segmentation

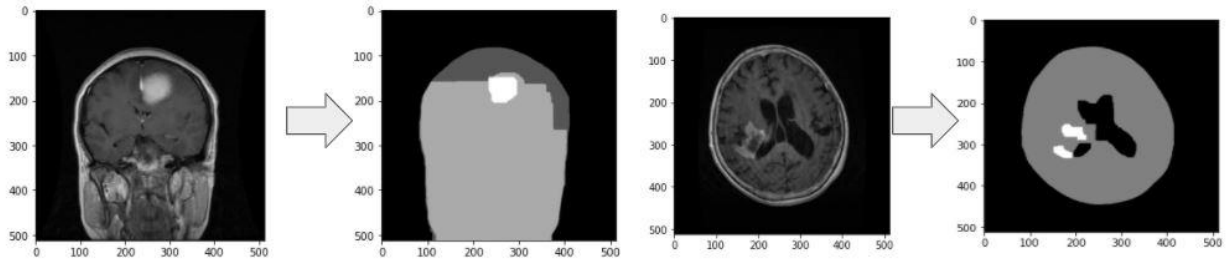
- Image Preprocessing
- Otsu Binarization for segmentation
- K-means clustering
- Applying colorform
- Created clusters
- Created blank cell array to store score
- Created Label

Results:



Graph cut

Results:



Note: As the dataset used is not having ground truth results cannot be compared so for classification decided to opt deep learning methods directly on dataset.

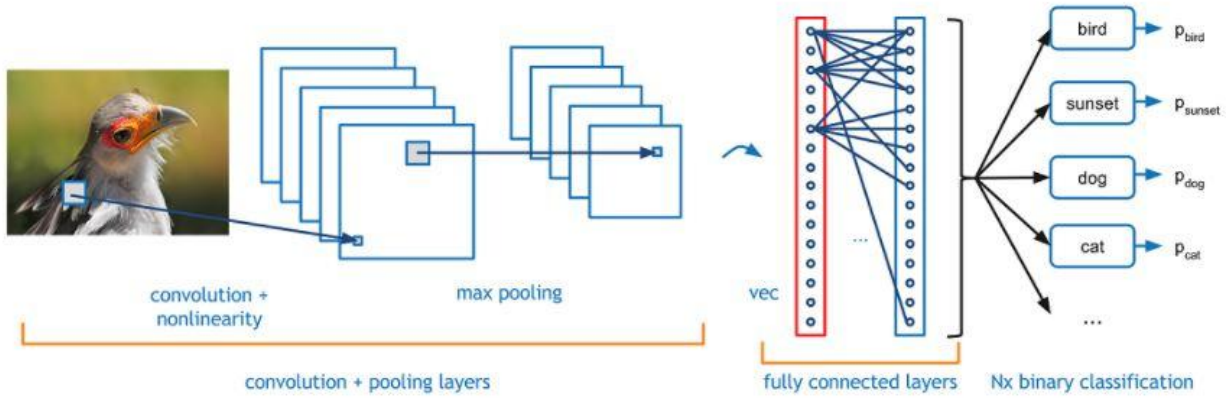
Convolutional Neural Network (CNN)

*In deep learning, a **convolutional neural network** (**CNN**, or **ConvNet**) is a class of artificial neural network, most commonly applied to analyze visual imagery.[wikipedia]*
It consists of multilayer with fully connected neural networks i.e. each neuron is connected to the other one in the next layer.This phenomenon of Fully connectivity many times leads to the overfitting.This overfitting can be avoided by penalizing parameters during training. In CNN different small and simple hierarchical patterns are used to mapping the complex features of images.

Steps in CNN

Step 1: Convolution

First building block is convolution.In which feature detectors are assigned, which basically serve as the neural network's filters.



source: https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

Step 2: Pooling

It is an accumulation of features from maps generated by convolving a filter over an image.

Step 3: Flattening

It is a breakdown of pooled layer into column matrix or 1D array for input to next layer

Step 4: Full Connection

Each cell is connected with each preceding cell and weight is given while training to classify the image

Image Preprocessing & Data preparation

- Resize all the images of dataset
- Splitted training and validation set in 80%-20%
- Defined epochs and batch size
- After that data is generated using `ImageDataGenerator function`

Models Used

For both the models batch size used is 32 and epochs is 30

Model 1: Non-regularized convolution

```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=3, padding= 'Same', activation='relu',input_shape=(X_train.shape[1],X_train.shape[2],1)
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters=128, kernel_size=3, padding= 'Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters=128, kernel_size=3, padding= 'Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters=256, kernel_size=3, padding= 'Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters=256, kernel_size=3, padding= 'Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

model.add(Conv2D(filters=512, kernel_size=3, padding= 'Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))

#FLATTENING

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

model.add(Dense(512, activation = "relu"))

model.add(Dense(4, activation = "sigmoid"))

#optimizer Used

model.compile(optimizer = 'SGD', loss = "categorical_crossentropy", metrics=["accuracy"])
```

Model 2: Regularized convolution

```
model1 = Sequential()
model1.add(Conv2D(filters=64, kernel_size=3, padding= 'Same', activation='relu', input_shape=(X_train.shape[1],X_train.shape[2],
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=128, kernel_size=3, padding= 'Same', activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=128, kernel_size=3, padding= 'Same', activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=256, kernel_size=3, padding= 'Same', activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=256, kernel_size=3, padding= 'Same', activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Conv2D(filters=512, kernel_size=3, padding= 'Same', activation='relu'))
model1.add(MaxPool2D(pool_size=(2,2)))
model1.add(BatchNormalization())

model1.add(Flatten())
model1.add(Dense(1024, activation = "relu"))
model1.add(Dropout(0.2))

model1.add(Dense(512, activation = "relu"))
model1.add(Dropout(0.2))

model1.add(Dense(4, activation = "softmax"))

model1.compile(optimizer = 'SGD' , loss = "categorical_crossentropy", metrics=["accuracy"])
```

Functions used in models:

SGD:

Stochastic gradient descent is the gradient descent optimization algorithm.
Algorithm:

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.

---

source : wikipedia

Softmax activation function

Formula

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

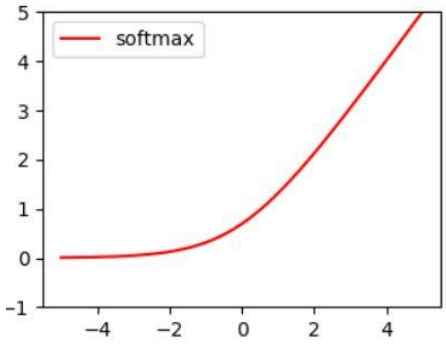
$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

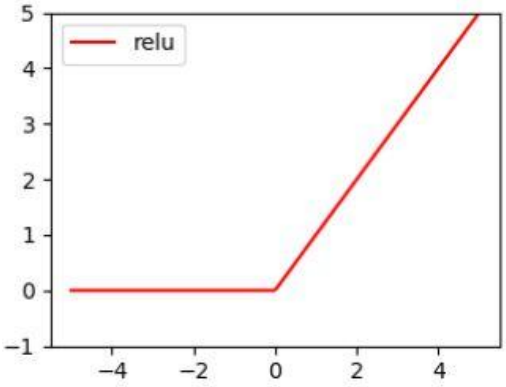
e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector



Relu activation function

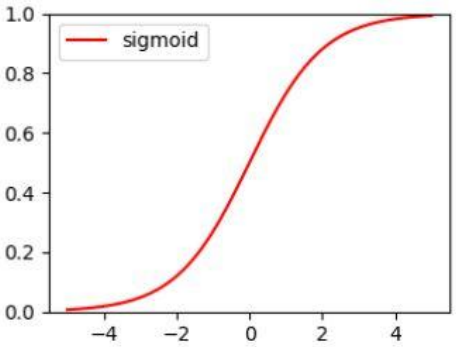
It is a piecewise linear activation function that gives out only if the input is positive otherwise it gives zero output.



Sigmoid activation function

Sigmoid function is used to get output in the range of [0,1]. It is usually suitable for finding the probability.

$$S(x) = \frac{1}{1 + e^{-x}}$$



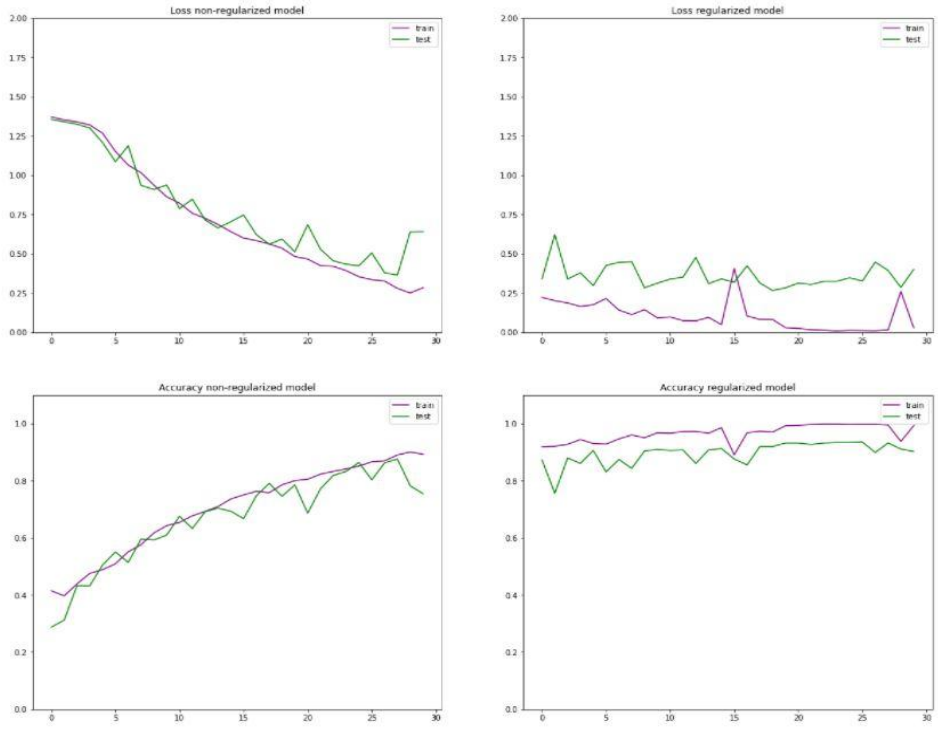
source:https://www.programmersought.com/article/12264689532/

Results:

Comparison of model 1 & model 2

Parameter	Model 1	Model 2
Normalization used	No	Yes
Batch size	32	32
epochs	30	30
Last activation function used	sigmoid	softmax
Validation Accuracy @epoch30	0.78	0.90
Validation loss @epoch30	0.63	0.39

Accuracy and loss variation w.r.t epochs

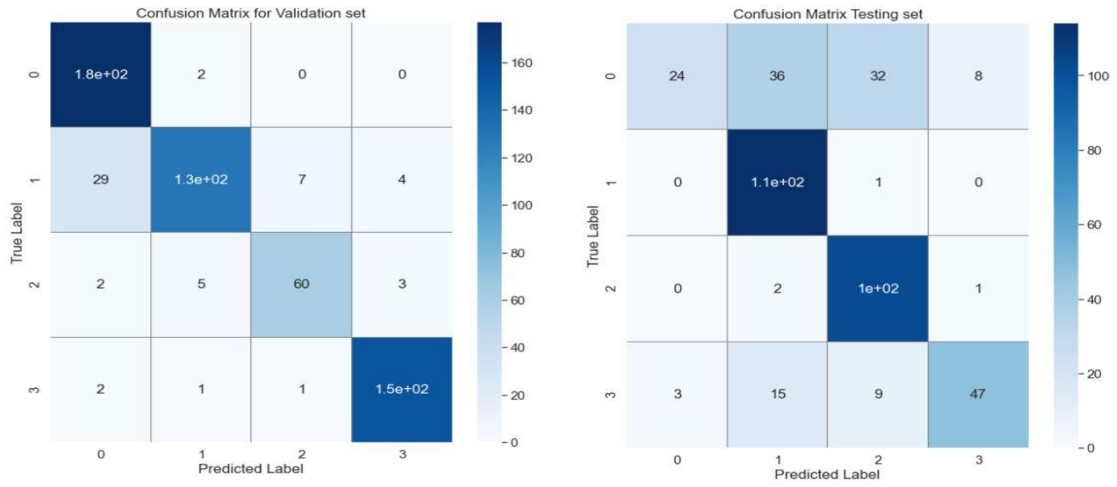


Conclusion: From these graphs we can conclude that the regularization is effective in this dataset as it is reducing overfitting and also increasing accuracy in both the training and validation set

Classification report:

	precision	recall	f1-score	support
0	0.84	0.99	0.91	179
1	0.94	0.76	0.84	169
2	0.88	0.86	0.87	70
3	0.96	0.97	0.97	156
accuracy			0.90	574
macro avg	0.91	0.90	0.90	574
weighted avg	0.91	0.90	0.90	574

Confusion matrices on validation and test set:



From literature we know the nature of a good confusion matrix, it must be as diagonally concentrated as possible and here in both the validation and test set it is more diagonally concentrated.

Transfer Learning Approach

What Is Transfer Learning?

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

Transfer learning has the advantage of decreasing the training time for a learning model and can result in lower generalization error.

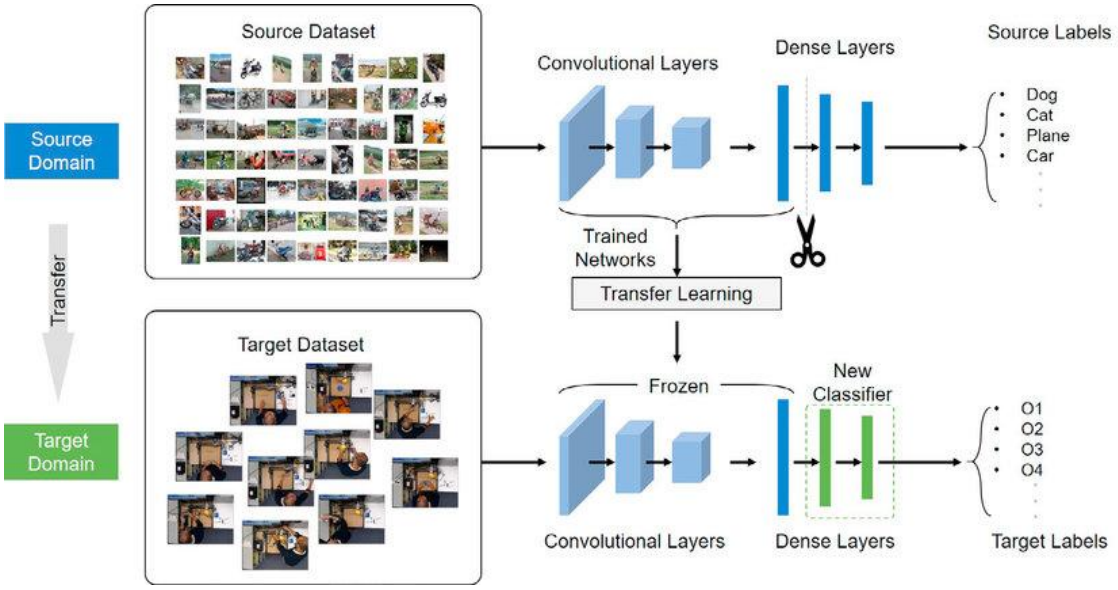
Also transfer learning is very useful when we have less data so we use pretrained weights from some pretrained network and add a few new layers to classify our dataset.

In case of image processing we use some pretrained models like VGG-16,RasNet,VGG-19,etc. which are trained on large ImageNet dataset.
source:https://analyticsindiamag.com/transfer-learning-for-multi-class-image-classification-using-deep-convolutional-neural-network/

What is imagenet?

The ImageNet project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images. source:https://en.wikipedia.org/wiki/ImageNet

Implementation of Transfer Learning can be best explained by following flow chart:



Models which we have used from keras are EfficientNetB0 and VGG16.

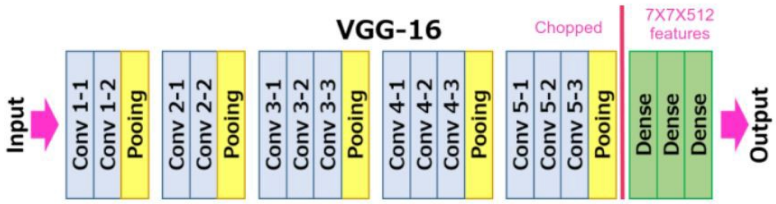
VGG16

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date.

Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output.

The 16 in VGG16 refers to it having 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

The following diagram best depicts how it is used for Transfer learning purpose



source:https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

EfficientNet

First introduced in Tan and Le, 2019 is among the most efficient models (i.e. requiring least FLOPS for inference) that reaches State-of-the-Art accuracy on both imagenet and common image classification transfer learning tasks.

The smallest base model is similar to MnasNet, which reached near-SOTA with a significantly smaller model. By introducing a heuristic way to scale the model, EfficientNet provides a family of models (B0 to B7) that represents a good combination of efficiency and accuracy on a variety of scales. Such a scaling heuristics (compound-scaling, details see Tan and Le, 2019) allows the efficiency-oriented base model (B0) to surpass models at every scale, while avoiding extensive grid-search of hyperparameters.

For B0 to B7 base models, the input shapes are different. For B0 it is (224,224,3)

source:https://keras.io/examples/vision/image_classification_efficientnet_fine_tuning/

Implementation and Results:

We tried implementingTransfer learning using both the models:

VGG16:

Models implemented for Transfer learning approach:
Model 1:

```
vgg_model = VGG16(weights='imagenet',include_top=False)
x=vgg_model.output
x=GlobalAveragePooling2D()(x)
x=Dense(128,activation='relu')(x)
x=Dropout(0.30)(x)
output=Dense(4,activation='softmax')(x)
model3=Model(inputs=vgg_model.input,outputs=output)#.to(device)
model3.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

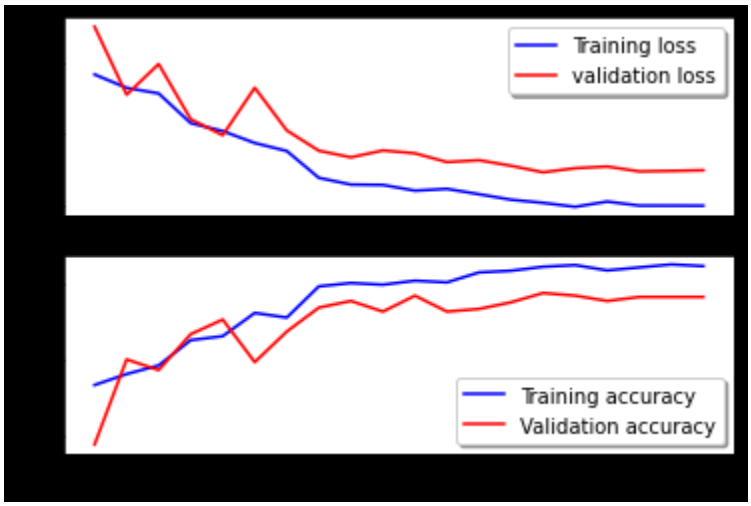
Model 2:

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
model = base_model.output
model = GlobalAveragePooling2D()(model)
model = Dense(4, activation='softmax')(model)
model = Model(inputs = base_model.input, outputs=model)
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

Best Results were obtained for Model 2

No. of epochs	20
Training Accuracy	92.38%
Training Loss	0.2009
Validation Accuracy	88.33%
Validation Loss	0.2996
Accuracy on Testing dataset	58%

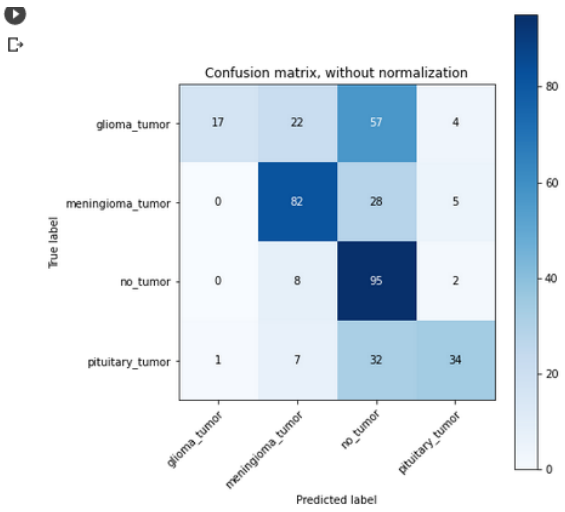
Loss and Accuracy Curves:



Metrics for VGG16 model:

	precision	recall	f1-score	support
0	0.94	0.17	0.29	100
1	0.69	0.71	0.70	115
2	0.45	0.90	0.60	105
3	0.76	0.46	0.57	74
accuracy			0.58	394
macro avg	0.71	0.56	0.54	394
weighted avg	0.70	0.58	0.54	394

Confusion Matrix:



To improve Testing and Validation accuracy we tried implementing other models in which EfficientNetB0 turned out to be best.

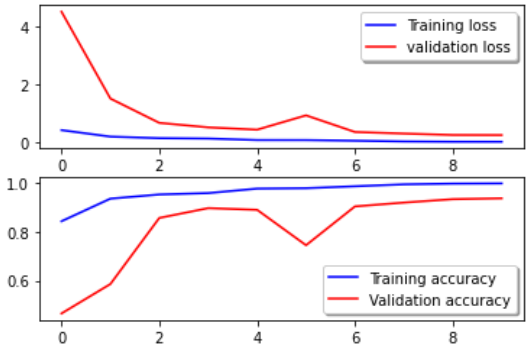
EfficientNetB0:

Model used :

```
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(image_size, image_size, 3))
model = base_model.output
model = GlobalAveragePooling2D()(model)
model = Dense(4, activation='softmax')(model)
model = Model(inputs = base_model.input, outputs=model)
model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

No. of epochs	10
Training Accuracy	99.65%
Training Loss	0.0123
Validation Accuracy	93.55%
Validation Loss	0.2447
Accuracy on Testing dataset	80%

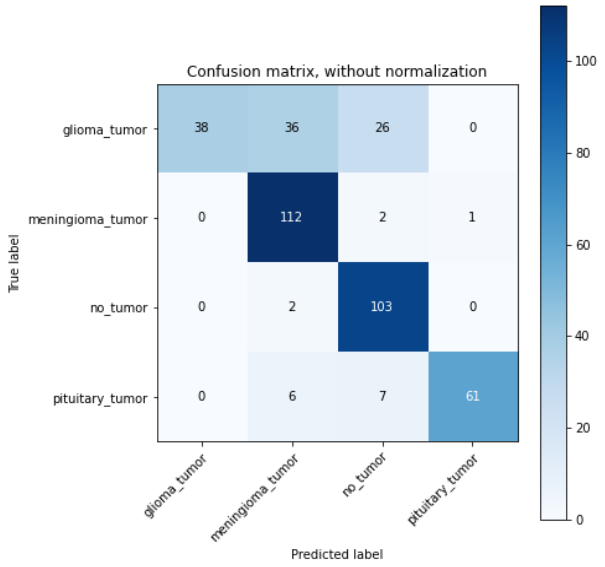
Loss and Accuracy Curves:



Metrics for EfficientNetB0 model:

	precision	recall	f1-score	support
0	1.00	0.38	0.55	100
1	0.72	0.97	0.83	115
2	0.75	0.98	0.85	105
3	0.98	0.82	0.90	74
accuracy			0.80	394
macro avg	0.86	0.79	0.78	394
weighted avg	0.85	0.80	0.78	394

Confusion Matrix:



Conclusion:

- Training and validation accuracy is far better in case of Transfer Learning approach 99.65% and 93.55% respectively. One reason can be for less data using pretrained weights through transfer learning helps network learn more.
- Time required for Transfer Learning models is considerably low.
- Our transfer learning model is working better on test dataset also with accuracy around 80% which was not the case in CNN.
- In both transfer learning approaches EfficientNetB0 worked better for us as we were getting less testing accuracy in case of VGG16.

Future Scope:

- One thing which can be implemented further is getting the survival rate of the patient by size or other features of the tumour.
- Early detection of brain tumor is possible by implementing accurate model at the time of MR Imaging.
- Also improving accuracy and optimising the network for time reduction can be done.

Contributions:

Initial data generation and classical segmentation : Both

CNN implementation: Piyush

Transfer learning approach: Shital

References:

1.Saad Albawi; Tareq Abed Mohammed; Saad Al-Zawi " Understanding of a convolutional neural network",DOI: 10.1109/ICEngTechnol.2017.8308186

2.Zar Nawab Khan Swati, Qinghua Zhaoc, Muhammad Kabira, Farman Alia, Zakir Alia,Saeed Ahmeda, Jianfeng Lua,"Brain tumor classification for MR images using transfer learning and fine-tuning",DOI:https://doi.org/10.1016/j.compmedimag.2019.05.001

3.https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef

4.https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/

5.https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

6.https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751