Final Project
Deep Learning

Isha Garg, Mrinalini Singh
NYU Tandon School of Engineering
Fall 2021
December 20th, 2021

Introduction

About one in six people suffer from central nervous system (CNS) disorders such as brain tumors, making it the 10th leading cause of death worldwide. A brain tumor is a cluster or mass of abnormal cells in the brain which can result in life-threatening increased pressure inside the skull. Early and accurate diagnosis of such a condition plays a key role in providing life-saving treatments for patients. While the 5-year survival rate for brain tumors is about 36 percent, survival heavily depends on age and type of tumor. There are over 120 different types of brain tumors that may arise in any region of the brain, brain stem or skull. These tumors may be benign (non-cancerous) or malignant (cancerous). Further, brain tumors may originate in the brain (primary) or have travelled to the brain from other organs such as breasts, colon, and kidneys (metastatic). Cancerous tumors are further categorized into stages I-IV based on disease progression and severity.

Evidently, identification and classification of brain tumors into different types based on location, origin, size, metastasis, and stage during diagnosis is vital. Doctors use Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) scans to visualize the tumor. However, this process is susceptible to human error and can take too inefficient under busy, stressful or time sensitive situations. Sometimes, biopsies are also used to aid in diagnosis by examining the tissue itself through surgical procedures. Unfortunately, biopsies pose a high risk to health of the patient and may be life-threatening.

To this end, deep learning can help reduce these problems. The development of deep learning tools in image analysis has created a huge potential for its use in medical imaging and diagnoses. Not only brain tumors, it can be applied to the characterization of other various disease conditions. The use of artificial intelligence can help doctors improve efficiency and accuracy of identification and classification of tumors from MRI and CT scans in a non-invasive manner that does not create additional health risks for the patient.

In this project, (1) classification of (MRI) scans to identify the presence and type of three different types of brain tumors based on location– Glioma, Meningioma, and Pituitary tumor will be explored. Gliomas account for 33 percent of all brain tumors and originate within the glial cells of the brain, Meningiomas form in the membranes surrounding the brain (meninges), and Pituitary tumors are an abnormal growth of the Pituitary gland present in the brain. To achieve this task, we have presented three different model approaches. Our first approach described is a convolution neural network model. To improve the results further, we also used transfer learning with two well-known pre-trained image classification models available– resnet18 and VGG16. Further, attempts were made to identify the location and boundary of tumor using segmentation methods but due to the lack of ground truth images in the dataset, we have limited the purpose of this project to classification only.

The dataset used contains 3264 images split into training (2870) and testing (394) sets and further divided into four class folders– Glioma, Meningioma, Pituitary tumor, and no tumor. The tumors in MRI scans appear brighter compared to the normal tissue. However, in order to accurately classify the images, several pre-processing techniques are required.

Challenges that occur during automated analysis of MRI like varying range and contrast of intensity, inhomogeneity, and noise can be tackled using image pre-processing to make the images comparable. Common pre-processing techniques include cropping to remove irrelevant part of the image and registration to align the images to a common anatomical space. It standardizes the MR images on a stereotaxic space. The aim is to get a multi-channel representation of each location in the brain by aligning the images of distinct sequences. Skull Stripping may be used where the skull in removed from images so that complete focus is laid on intracranial tissues.There are different procedures such as Brain Extraction Tool (BET), Robust Brain Extraction (Robex) etc. Bias Field Correction corrects the variations in image contrast that occur due to magnetic field inhomogeneity. The common approach in this case is N4 bias field correction. Intensity Normalization is extremely common for any image classification task where intensities of the images is mapped to standard scale using the mean and standard deviation across all channels. Here, image intensities are piecewise linearly mapped to a scale and z-score are computed where mean image intensity is subtracted from the pixels in the image and further, they are divided by S.D.(Standard Deviation) of intensities. Noise Reduction: It is the reduction of

noise in the detected in MR images. Image data augmentation can help is useful for preventing overfitting especially in small data sets. Image data augmentation can be used to artificially increase the size of the training set by creating modified versions of the given images. An important advantage of this method is that it makes the model more sensitive to slight variations and hence, less prone to overfitting.

After pre-processing, the data can be loaded onto the model for training. The model chosen for this image classification problem is convolutional neural networks (CNN) mainly because of its advantages of lower computational cost, spatial relationships, and overcoming confounding features compared to fully connected neural networks. Additionally, training the networks with transfer learning from pre-trained neural architectures is usually much faster and easier than training the networks with randomly initialized weights. Since the dataset is small, the accuracy of the model may not be very good and hence, transfer learning will also be used to give improved results and prevent overfitting.

Cross Entropy Loss, a widely used loss function that can calculate the difference between the probability distribution of the labels and predictions, will be used for this task. Further, stochastic gradient descent (SGD) is a very popular optimizer in machine learning and has been successfully implemented in many large-scale problems. It is efficient and easy to implement. Computing the gradient of the loss function with respect to the parameters for the whole training set is not feasible due to the large number of parameters involved. Instead, SGD updates all the parameters for each training data point and the output individually which greatly reduces the computational cost. However, SGD is noisier and requires a higher number of iterations to reach the minima because of the randomness in its descent. Further SGD has challenges with respect to sparse data where some features are frequently occurring and others are rare. In such a case, having the same learning rate will not give good results since rare features would need a larger update than frequently occurring features. To tackle this challenge, Adaptive Moment Estimation or ADAM is a variation of SGD that able to adapt the learning rate to the parameters. In this project, we will explore and compare both SGD and ADAM in our models.

Literature Survey

There has been a development of various types of deep learning approaches for distinct purposes such as segmentation and object detection in images, genotype/phenotype detection, speech recognition and classification of diseases. Common ones of the lot are Deep Boltzmann machines, stacked auto-encoders, Deep neural networks, and Convolutional neural networks (CNNs) and of these, CNN is most suitable for image segmentation and classification. Segmentation of brain tissues, these days is also done through CNN in which explicit definition of intensity and spatial features is avoided.

In their paper, Shree et.al. (2018) have focused on noise removal techniques, extraction of gray-level co-occurence matrix (GLCM) features, and discrete wavelet transform (DWT)-based brain tumor region growing segmentation to improve performance. Pre-processing steps included signal-to-noise ratio improvement and smoothening. They used a probabilistic neural network classifier to train and test the performance accuracy and achieved nearly 100 percent accuracy in identifying tumors.

Irmak et.al. (2021) proposed three CNN models to classify brain tumors. An accuracy of 99.33 percent was achieved for brain tumor detection using their first model. The second model classified the tumor into five types and achieved an accuracy of 92.66 percent. The third model classified the tumors into three grades with an accuracy of 98.14 percent. Since CNN models require multiple architectural (filter, stride, padding, etc.) and fine-tune (l_2 regularization, batch size, learning rate) hyper-parameters, the authors used two grid search optimization algorithms for selecting the best hyper-parameters. The grid search was performed on the training set with a five-fold cross-validation procedure.

Diaz-Pernes et.al. (2021) presented a CNN model with a multi-scale approach for classification of three types brain tumors while simultaneously also performing segmentation in the absence of image pre-processing that removes skull and vertebral column parts. An accuracy of 97.3 percent was achieved. Their multi-pathway CNN architecture processes the MRI image pixel by pixel using a sliding window to classify each pixel into one of four labels (healthy, meningioma, glioma, and pituitary).

Sajjad et.al. (2019) performed segmentation and classification on brain tumor MRIs using CNN. They also compared the effect of data augmentation on the accuracy of the model. The augmentation techniques included rotations, flips, Gaussian blur, sharpness, edge detection, emboss, skewness, and shear rotations. After performing augmentation, their model accuracy increased from 87.38 percent to 90.67 percent. Further the overall sensitivity increased from 84.51 percent to 88.41 percent and overall specificity increased from 93.34 percent to 96.12 percent.

Chelghoum et.al. (2020) proposed an automatic classification system designed for three types of brain tumors (glioma, meningioma and pituitary tumor) using CE-MRI images. Transfer learning was applied to nine pre-trained architectures (AlexNet, GoogleNet, VGG16, VGG19, ResNet18, ResNet50, ResNet101, ResNet-Inception-v2 and SENet) for brain MRI images classification trained for three epochs. ztheir method involved modifying the last three layers of the pre-trained networks, replacing the fully conncted layer to another one that represents the classification task and finally using the model for MRI data. VGG-16 achieved the highest accuracy of 98.7 percent with lowest number of epochs in limited time.

Reddy et.al. (2018) proposed work for detecting brain tumor by image segmentation using filtering, K-means clustering, and thresholding for skull removal, segmentation and performance measures. First, median filter was used to reduce noise and provide a normalized intensity in the overall image. Using K-means clustering, the regions with similar intensity was clustered using a K-means algorithm. The image pixels with the maximum intensity were identified to be tumor portions and the threshold value was adjusted to be 30 pixels lower than the maximum pixel.

Model description

Input for CNN Model:

Dataset: 3264 RGB 150x150 images of brain MRI scans

- Training set: 2583 images
- Validation set: 287 images
- Testing set: 394 images

Input for Transfer Learning Model:

Dataset: 3264 RGB 150x150 images of brain MRI scans

- Training set: 2870 images
- Testing set: 394 images

Pre-processing:

- Normalization: Mean and standard deviation of each of the three RGB channels across all images in the set was found.
- Data Augmentation for CNN Model(Tensorflow): Random horizontal flip, random vertical flip, random rotation, and shifts in brightness were performed to introduce variations in the images and make the model more robust towards such changes using Image Data Generator.
- Data Augmentation for Transfer Learning Model(Pytorch): Random horizontal flip was performed to introduce variations in the images and make the model more robust towards such changes using Data Transforms.

Model 1: CNN

The convolutional neural network model includes batch normalization, three (3x3) convolution layers, each followed by ReLU activation functions and (2x2) MaxPooling layers. The output is added to a dropout layer, two linear layers and finally a softmax activation function that will produce probabilities for each of the four classes.

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization_8 (Batc  (None, 150, 150, 3)      12
 hNormalization)

 conv2d_37 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_37 (MaxPoolin  (None, 74, 74, 32)       0
 g2D)

 conv2d_38 (Conv2D)          (None, 74, 74, 32)        9248

 max_pooling2d_38 (MaxPoolin  (None, 37, 37, 32)       0
 g2D)

 conv2d_39 (Conv2D)          (None, 37, 37, 64)        18496

 max_pooling2d_39 (MaxPoolin  (None, 18, 18, 64)       0
 g2D)

 conv2d_40 (Conv2D)          (None, 18, 18, 128)       73856

 max_pooling2d_40 (MaxPoolin  (None, 9, 9, 128)        0
 g2D)

 dropout_7 (Dropout)         (None, 9, 9, 128)         0

 flatten_9 (Flatten)         (None, 10368)             0

 dense_18 (Dense)            (None, 128)               1327232

 dense_19 (Dense)            (None, 4)                 516

 activation_8 (Activation)   (None, 4)                 0

=================================================================
Total params: 1,430,256
Trainable params: 1,430,250
Non-trainable params: 6
```

Model 2: Transfer Learning with VGG16

VGG16 is a popular pretrained model with the following layers as shown. Only the last five layers have been modified to use for our dataset. These layers include linear layer, ReLU activation, Dropout, second linear layer and finally, a log softmax layer that gives the output.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Sequential(
      (0): Linear(in_features=4096, out_features=256, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
      (3): Linear(in_features=256, out_features=4, bias=True)
      (4): LogSoftmax(dim=1)
    )
  )
)
```

Model 3: Transfer Learning with resnet18

The second transfer learning model was done using the resnet18 pre-trained model. The layers of the resnet18 model are shown below. Only a linear layer was added as the last layer gave an output with 4 dimensions for the 4 classes present.

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Loss function:

Cross Entropy Loss - We used Cross Entropy Loss as it provides the best result when have to adjust the model weights during training.

Optimization:

ADAM and Stochastic gradient descent (SGD)

Output:

Classification of images into one of four classes– (1) No tumor, (2) Glioma tumor, (3) Meningioma tumor, and (4) Pituitary tumor.

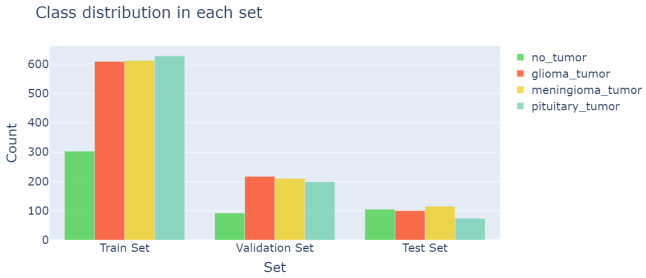
Evaluation:

The model's performances were measured using accuracy and loss curves in addition to confusion matrices. The CNN model was trained on the training and validation set and evaluated on the testing set. The transfer learning models were trained and evaluated using only the training and testing sets.

Results

Model 1: CNN

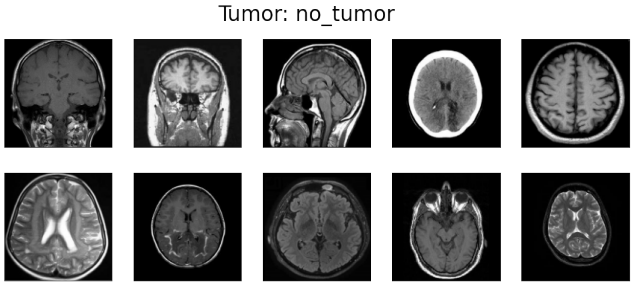
Distribution of classes:

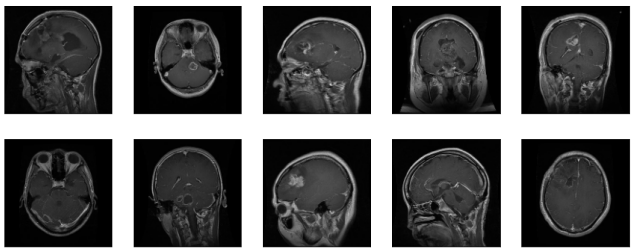


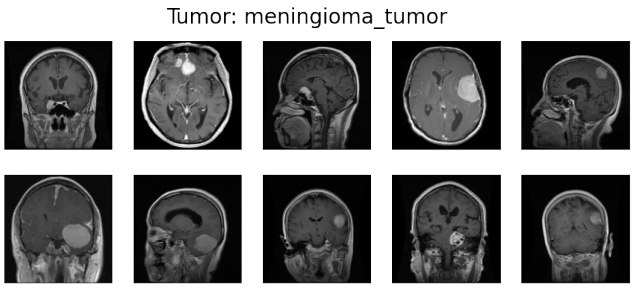
The figure given shows the frequency of the four classes– No tumor, glioma tumor, meningioma tumor, and pituitary tumor present in the training, testing, and validation sets. The training set contains 2152 images, the testing set contains 394 images and the validation set contains 718 images.

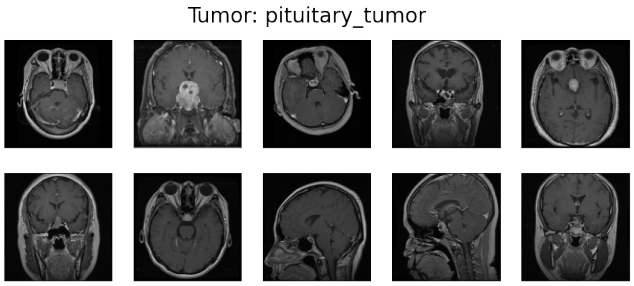
Sample visualization:

The figure shows a sample of 10 images within each class to visualize what the dataset contains. Evidently, tumors are occurring with different sizes, in different locations, and appear brighter than the normal tissue surrounding it. It can also be seen that some images are brighter than others and contain MRIs taken from different brain orientations. Hence, normalization and augmentation is being used to make the model more robust.

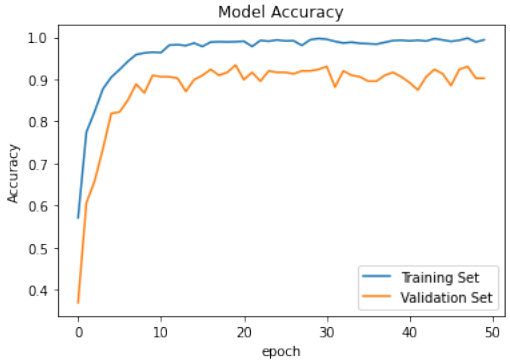


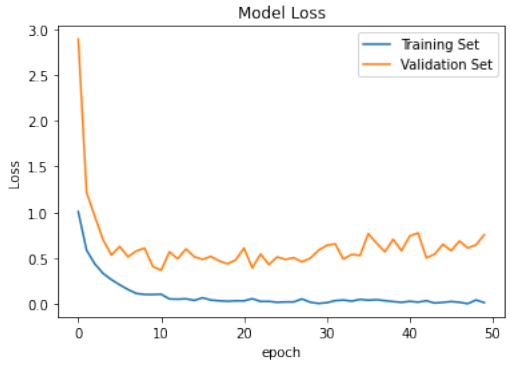




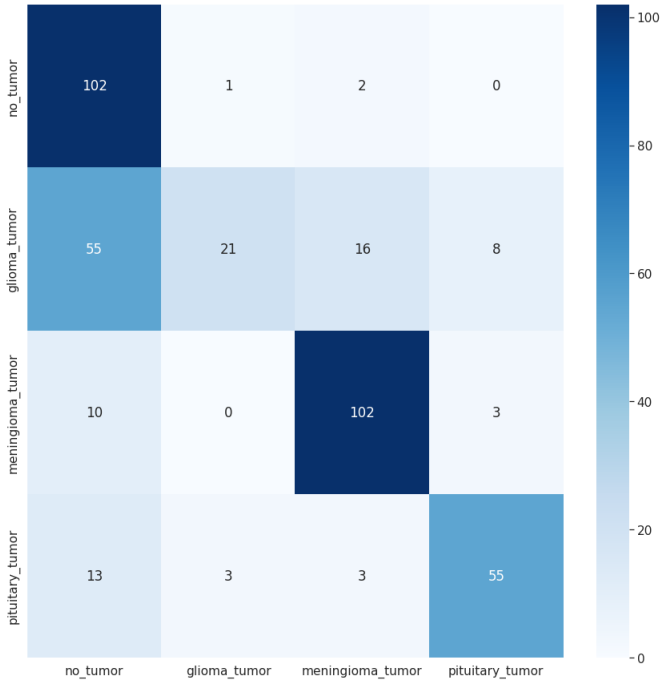


The CNN model was run for 50 Epochs and achieved a high training accuracy of over 99 percent and highest validation accuracy of 94.77 percent. The testing accuracy was 73.86 percent. The accuracy and loss curves are shown in the figures below.





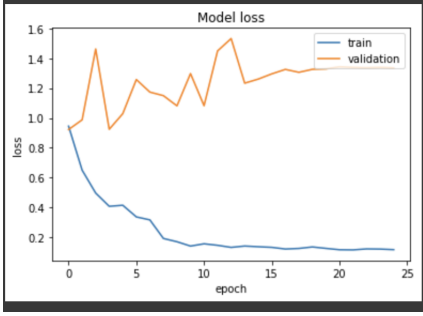
The confusion matrix for the CNN model is shown below. Most errors occurred as the glioma tumor was wrongly predicted under no tumor.



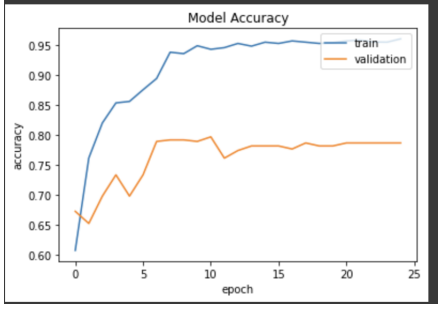
Model 2: Transfer Learning with VGG16

Below is the accuracy curve, loss curve and confusion matrix plotted for VGG16 model

```
# ------------ Plot loss and Acc curves
plt.plot(model_tr_loss, label = 'train')
plt.plot(model_val_loss, label = 'validation')
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```

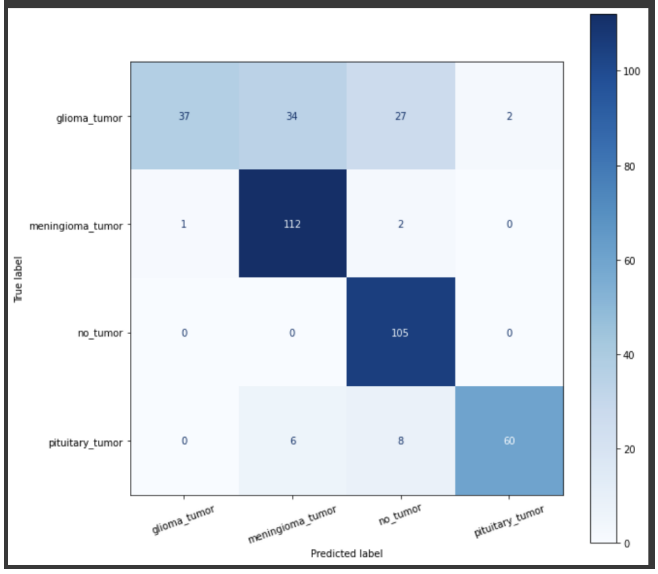



```
# ------------ Plot loss and Acc curves
plt.plot(model_tr_acc, label = 'train')
plt.plot(model_val_acc, label = 'validation')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```



The confusion matrix for the VGG16 model is shown below. Most errors occurred as the glioma tumor was wrongly predicted under meningioma tumor and no tumor

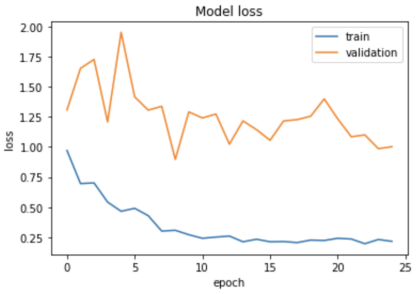
```
plot_confusion_matrix(labels, probs, class_names)
```



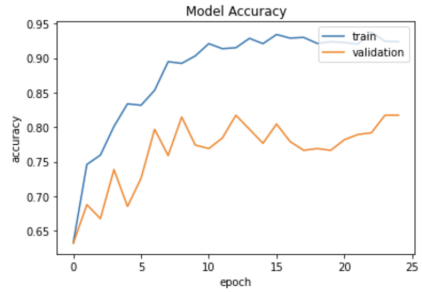
Model 3: Transfer Learning with resnet18

Below is the accuracy curve, loss curve and confusion matrix plotted for VGG16 model

```
# ------------ Plot loss and Acc curves
plt.plot(model_tr_loss_v, label = 'train')
plt.plot(model_val_loss_v, label = 'validation')
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```



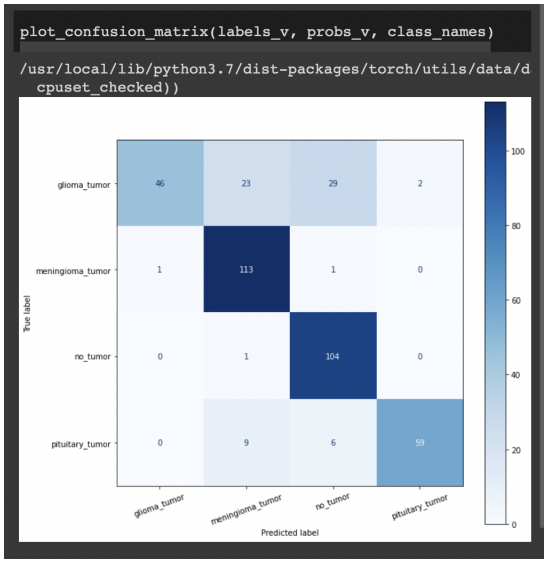

```
# ------------ Plot loss and Acc curves
plt.plot(model_tr_acc_v, label = 'train')
plt.plot(model_val_acc_v, label = 'validation')
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(loc='upper right')
plt.show()
```



The confusion matrix for the Resnet18 model is shown below. Most errors occurred as the glioma tumor was wrongly predicted under meningioma tumor and no tumor.

```
plot_confusion_matrix(labels_v, probs_v, class_names)
```

```
/usr/local/lib/python3.7/dist-packages/torch/utils/data/d
  cpuset_checked))
```



Conclusion

The table below shows a comparison of the accuracies of each model.

Model	CNN	Transfer Learning – Resnet 18	Transfer Learning – VGG16
Training Accuracy	>99%	92.37%	~96%
Testing Accuracy	73.86%	81.7%	79.6%
Optimizer	Adam	SGD	SGD

In this project, our goal was to perform classification on brain MRI images into four different classes– No tumor, Glioma tumor, Meningioma tumor, and Pituitary tumor. We have also pre-processed images using cropping, median filter, K Means clustering, and thresholding for Segmentation. However, since we do not have ground truth images, we did not train the images for segmentation.

We ran 3 different classification models on the dataset and have tabulated the results. We can see that we achieved an accuracy in the range of 70-75 percent for the CNN model and the Training accuracy achieved is almost 100 percent for 50 Epochs which is quite satisfactory. For the Transfer Learning models, our accuracy has spiked up to 82 percent for Resnet 18 with a training accuracy of 92.37 percent. For VGG16, the testing and training accuracy is 80 and 96 percent respectively. Both the transfer learning models have been run for 25 Epochs.

Highest errors in our all 3 models was due to misclassification of glioma tumor to no tumor for CNN and to meningioma tumor and no tumor in Transfer Learning Models. Glioma tumors also seem to appear very small in the samples visualized. Hence, in future work, accuracy results may be improved by focusing on better identifying glioma tumors.

Supplementary material: Segmentation

Model description

In addition to classification, segmentation of the brain tumor can further aid doctors in the diagnosis of central nervous

system disorders and optimize treatment plans for the patient. In addition to classification, we developed a plan to perform segmentation of the brain tumors in the dataset using work done by Reddy et.al. as guide. Due to the lack of ground truth images, segmentation could not be perfomed but we have included the pre-processing step taken:

- Cropping: Tee images can be cropped to only to remove some background using thresholding and contours.
- Median filter: The median filter parses through the image pixel by pixel, replacing each one with the median of its neighbouring pixels. Hence, it provides a normalized intensity for the overall image.
- K-means Clustering: In K-means clustering, regions with similar intensities are clustered together. We chose to have three clusters to differentiate the background, normal tissue, and tumor. In the K Means approach, centers are allocated to an arbitrary value of k. All the points closest to the each center is marked and the mean for each cluster is determined. This process is repeated until the mean does not change any further.
- Thresholding: Based on a chosen threshold, each pixel is assigned a value of 0 if it is below the threshold or a value of 1 if it is above the threshold. Since the tumor has a higher intensity, it can be differentiated from the normal tissue this way.
- Skull removal: The skull appears in the images with high and varying intensities. Hence, it can interfere with the segmentation of the tumor and should be removed. Morphological operations like erosion, dilate, close, and open can be used to discard the skull without damaging the tumor regions.
- Segmentation: Segmentation can then be performed using a pre-trained UNet model with transfer learning.
- Evaluation: Segmentation can be evaluated using accuracy, loss curves and confusion matrices if the ground truth images are provided.

Results

Original image:

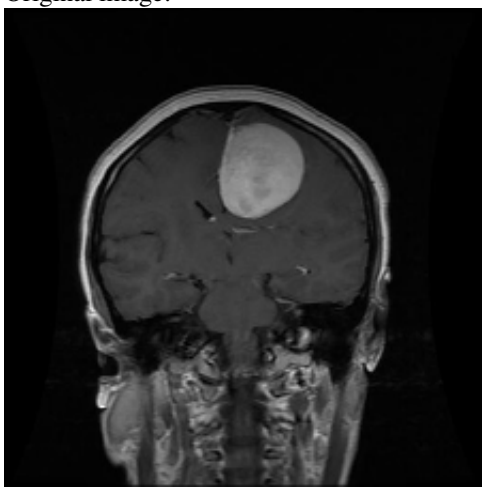


Image after cropping:

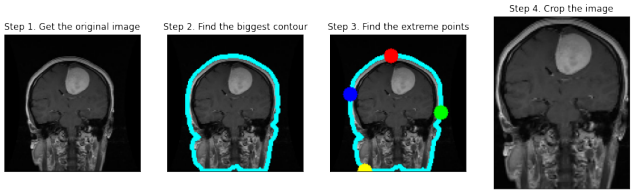


Image after median filter:

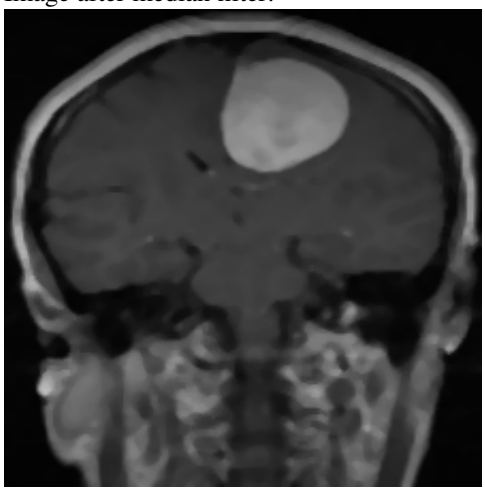


Image after kmeans clustering:

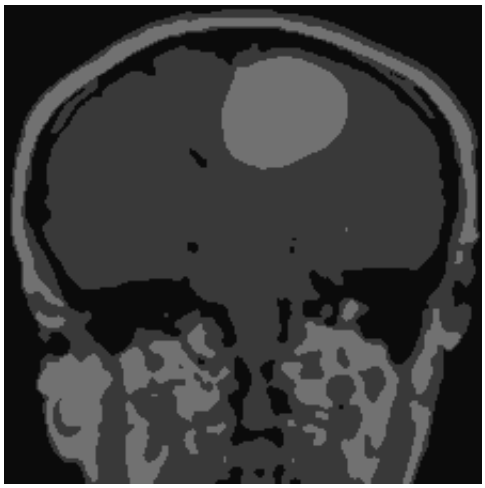


Image after thresholding:

Source code

Link: https://github.com/Singh-Mrinalini/Brain-MRI-Classification-CNN-

References

- Dataset source link: https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri
- https://chinmayhegde.github.io/dl-notes/
- http://d2l.ai/chapter$_{c}onvolutional - neural - networks/index.html$
- Varuna Shree, N., Kumar, T.N.R. Identification and classification of brain tumor MRI images with feature extraction using DWT and probabilistic neural network. Brain Inf. 5, 23–30 (2018). https://doi.org/10.1007/s40708-017-0075-5
- Irmak, E. Multi-Classification of Brain Tumor MRI Images Using Deep Convolutional Neural Network with Fully Optimized Framework. Iran J Sci Technol Trans Electr Eng 45, 1015–1036 (2021). https://doi.org/10.1007/s40998-021-00426-9
- Díaz-Pernas, F. J., Martínez-Zarzuela, M., Antón-Rodríguez, M., González-Ortega, D. (2021). A Deep Learning Approach for Brain Tumor Classification and Segmentation Using a Multiscale Convolutional Neural Network. Healthcare (Basel, Switzerland), 9(2), 153. https://doi.org/10.3390/healthcare9020153
- Chelghoum, R., Ikhlef, A., Hameurlaine, A., amp; Jacquir, S. (2020). Transfer learning using convolutional neural network architectures for brain tumor classification from MRI images. IFIP Advances in Information and Communication Technology, 189–200. https://doi.org/10.1007/978-3-030-49161-$1_{1}7$
- D. Reddy, Dheeraj, Kiran, V. Bhavana and H. K. Krishnappa, "Brain Tumor Detection Using Image Segmentation Techniques," 2018 International Conference on Communication and Signal Processing (ICCSP), 2018, pp. 0018-0022, doi: 10.1109/ICCSP.2018.8524235.
- https://pytorch.org/tutorials/beginner/transfer$_{l}earning_{t}utorial.html$
- Sudre, C. H., Li, W., Vercauteren, T., Ourselin, S., amp; Jorge Cardoso, M. (2017). Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support, 240–248. https://doi.org/10.1007/978-3-319-67558-$9_{2}8$
- Sille, R., Choudhury, T., Chauhan, P., amp; Sharma, D. (2021). Dense hierarchical CNN – a unified approach for Brain Tumor Segmentation. Revue D'Intelligence Artificielle, 35(3), 223–233. https://doi.org/10.18280/ria.350306
- Muhammad Sajjad, Salman Khan, Khan Muhammad, Wanqing Wu, Amin Ullah, Sung Wook Baik. Multi-grade brain tumor classification using deep CNN with extensive data augmentation. Journal of Computational Science, Volume 30, 2019. Pages 174-182. ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2018.12.003