Using a pre-trained CNN to classify MRI images into 44 classes

Vinay Bhandaru

DATS 6303

Spring 2023

Introduction

The topic for this final project is classifying MRI images of brain tumors found on the Kaggle website (1). The topic for this final project is classifying MRI images of brain tumors found on the Kaggle website (1). This report begins with a description of the images and the 44 classes. It then discusses the Resnet deep learning neural network and how it will be trained and evaluated. Finally, the results of the final network are explained and conclusions drawn.

Description of the data set

There are 4479 brain MRI images on the Kaggle website (1). The image files are downloadable as a zip file from Kaggle's website. All of the image file extensions are either .jpg or .jpeg with the exception of one .webp file. Using Python OpenCV to read in the images, all images have 3 channels. The length and width of the images vary. The first dimension ranges between 347 and 630 pixels and the second dimension ranges between 305 and 630 pixels. 2232 or approximately 50% of the images are read in with a dimension of 630X630.

The images don't have any patient identifiers or markings. The classification or label of an image is determined by the name of the directory in which it is located. There are 44 directories containing images corresponding to 44 classes. Classes are based on combinations of tumor type and image contrast type.

There are 3 types of images based on contrast, T1, T1C+, and T2. In T1-weighted MRI images the signal of the fatty tissue is enhanced and the signal of the water is suppressed. In T2-weighted MRI images the signal of the water is enhanced (2). T1C can stand for gadolinium-contrast-enhanced T1-

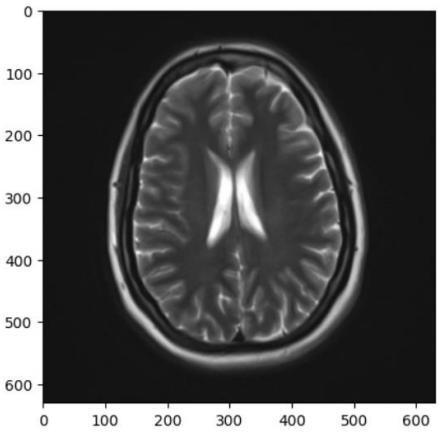
weighted MRI. One study found that classification was the best for T1, then T1C+, and then T2 images.

Of the 4479 MRI images only 30% are T2.

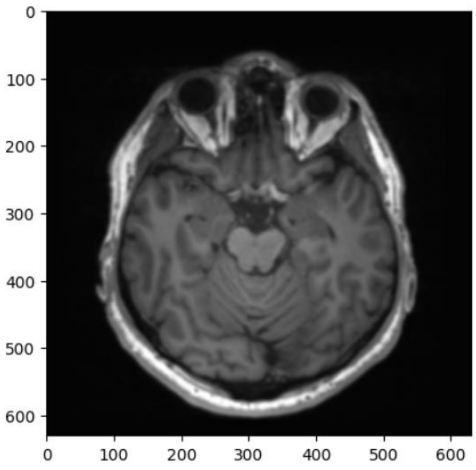
Table 1. Contrast Image Types

	n	%
T1C+	1693	37.8
T1	1423	31.8
T2	1363	30.4

Normal T2 image



Normal T1 image



Images are classified as either normal or as one of 14 types of brain tumors. These types are astrocytoma, carcinoma, ependymoma, ganglioglioma, germinoma, glioblastoma, granuloma, medulloblastoma, meningioma, neurocytoma, oligodendroglioma, papilloma, schwannoma, and tuberculoma. Each of the 14 brain tumor types have images of each type T1, T1C+, or T2. Normal images are either T1 or T2, and there are none of type T1C+. As a result there are 44 classes of images.

Table 2. Brain Tumor Type

	n	%
Normal	522	11.7
Meningioma	874	19.5
Astrocitoma	580	12.9
Schwannoma	465	10.4
Neurocitoma	457	10.2
Carcinoma	251	5.6
Papiloma	237	5.3
Oligodendroglioma	224	5
Glioblastoma	204	4.6
Ependimoma	150	3.3
Tuberculoma	145	3.2
Meduloblastoma	131	2.9
Germinoma	100	2.2
Granuloma	78	1.7
Ganglioglioma	61	1.4

11.7% of the images are normal and the remaining have a brain tumor based on interpretation of the images by radiologists. The most common type of brain tumor is Meningioma. 874 or nearly 20% of the images are categorized as Mengioma. A meningioma is a tumor that originates from the meninges, which are membranes that surround the brain and spinal cord (3). The class with the most number of images is Meningioma T1C+ and has 369 images. The class with the least number of images is Granuloma T2 and has 17 images.

Description of the deep learning network and training algorithm

The deep learning network that will be used to classify these images is ResNet-18 from Deep Residual Learning for Image Recognition (7). The pretrained model and weights are available through Python Pytorch. A complete description of the network can be found in the appendix. It consists of 20 convolutional and batch normalization layers in addition to a max pooling layer and an adaptive average pooling layer. The only change to the model is the final layer which is a fully connected layer of 44 neurons. This network requires images with 3 channels, since the first argument to the Conv2d function in the first layer is 3, which is the number of in channels. The loss function is the cross entropy loss and is shown below.

$$L_{\mathrm{CE}} = -\sum_{i=1}^{n} t_i \log(p_i), \quad \text{for n classes,}$$

In Pytorch the equation implemented is slightly different. The cross entropy loss is calculated between the target and the input logits for each class, which do not need to be positive or sum to 1 because the loss function will automatically do this in Pytorch.

The training algorithm that will be used is a variation of stochastic gradient descent, known as mini-batch gradient descent. In mini-batch gradient descent, the weights and biases are updated after computing the gradient of the loss for a subset of the training set or mini-batch.

Experimental Setup

The 4479 brain MRI images will be split into a training set, consisting of 70% of the images, a validation set consisting of 15% of the images, and a test set consisting of the remaining 15% of the images. The network will be trained on the training set and evaluated on the validation set to determine the best way to resize the images. Then the final model will be evaluated on the test set to determine how well it classifies the images.

The network will be implemented in Pytorch by defining a class based on NN.module, which is a base class for all neural network models. The Resnet-18 pre-trained model with the only change being to change the number of neurons in the output layer to 44, which is the number of image classes. The model will be trained for at least 5 epochs using minibatches. The size of the mini-batches will be as large as possible. The larger the re-sized image the smaller the mini-batch size due to memory restrictions. The optimization algorithm will be SGD and learning rate 0.01. Overfitting will be prevented by using the validation set to evaluate.

As mentioned previously, the dimensions of the input images vary, and the maximum size is 630x630. Resizing all images to a common size allows for training with mini-batches of any size. As shown by the equations below, precision measures how accurate positive predictions are, recall measures the sensitivity or what proportion of positives are accurately predicted, and F1 is a function of both precision and recall. These measures are calculated for each class and then macro-averaged, meaning all classes equally contribute to the final averaged metric regardless of the size of the class.

precision = TP/(TP+FP)

recall = TP/(TP+FN)

F1-score= (2*Precision*Recall)/(Precision+Recall)

Results

Table 4 shows the performance of 3 models on the validation dataset. As shown by the equations below, precision measures how accurate positive predictions are, recall measures the sensitivity or what proportion of positives are accurately predicted, and F1 is a function of both precision and recall. All of the averages shown in table 4 are macro-averaged, meaning all classes equally contribute to the final averaged metric regardless of the size of the class.

Table 3. Effect of resizing image on validation metrics

	Batch size	Training accuracy	Accuracy	Avg Precision	Avg Recall	Avg F1
100x100	50	0.998	0.689	0.72	0.59	0.62
256X256	25	0.998	0.768	0.81	0.67	0.71
630x630	15	0.978	0.83	0.76	0.75	0.75

The first model is based on reducing all images to 100X100. The overall accuracy on the validation dataset is 69%, and the macro-averaged F1 is 0.62. The batch size was 50. Reducing image size to 256x256 the validation set accuracy increased to 77% and the macro-averaged F1 to 0.71. The batch size was reduced to 25. Reducing image size to 630X630 the validation set accuracy increased to 83%. Although the macro averaged precision was lower for 630x630 compared to 256x256, the macro-averaged F1 increased to 0.75. Training was done with a batch size of 15. Based on these results the images were resized to 630x630. These models all achieved a 98% or higher accuracy on the training set, regardless of batch size, so these results are valid and 630x630 is better.

The pre-trained Resnet-18 network utilizing images of size 630X630 was evaluated on the test set. The test set consists of 15% of the images or 672 images. The overall accuracy is 83.6%, which almost the

same as the validation set accuracy. The macro-averaged F1 is 0.76, which is also almost the same as the validation set.

Table 4. Performance on test set

	Accuracy	Avg Precision	Avg Recall	Avg F1
	0.84			
Macro Average		0.8	0.75	0.76
Weighted Average		0.84	0.84	0.83

The table below shows the metrics for each individual class and is sorted by lowest F1 score to highest F1 score.

Table 5. Performance on Test Set

	precision	recall	f1score	support
Granuloma T1	0	0	0	5
Granuloma T2	0	0	0	2
Tuberculoma T2	0	0	0	5
Meduloblastoma T2	0.67	0.33	0.44	6
Glioblastoma T2	0.57	0.5	0.53	8
Ependimoma T2	0.44	0.88	0.58	8
Ependimoma T1	0.8	0.57	0.67	7
Ganglioglioma T1C+	0.67	0.67	0.67	3
Germinoma T1	1	0.5	0.67	4
Meduloblastoma T1	1	0.5	0.67	4
Tuberculoma T1C+	0.78	0.58	0.67	12
Papiloma T1	0.56	0.9	0.69	10
Granuloma T1C+	0.67	0.8	0.73	5
Schwannoma T1C+	0.68	0.79	0.73	29
Oligodendroglioma T2	0.62	1	0.77	10
Glioblastoma T1C+	1	0.64	0.78	14
Astrocitoma T2	0.86	0.73	0.79	26
Papiloma T1C+	0.92	0.69	0.79	16
Astrocitoma T1	0.71	0.92	0.8	26
Ganglioglioma T1	1	0.67	0.8	3
Ganglioglioma T2	1	0.67	0.8	3

Schwannoma T2	0.73	0.89	0.8	18
Ependimoma T1C+	1	0.71	0.83	7
Meningioma T2	0.81	0.86	0.83	35
Carcinoma T2	1	0.73	0.84	11
Neurocitoma T2	0.81	0.87	0.84	15
Schwannoma T1	0.83	0.86	0.84	22
_NORMAL T2	0.83	0.85	0.84	41
Meningioma T1	0.85	0.85	0.85	41
Meduloblastoma T1C+	0.82	0.9	0.86	10
Carcinoma T1C+	0.81	1	0.89	17
Germinoma T2	1	0.8	0.89	5
Glioblastoma T1	0.89	0.89	0.89	9
Papiloma T2	1	0.8	0.89	10
Astrocitoma T1C+	1	0.86	0.92	35
Meningioma T1C+	0.87	0.96	0.92	56
Carcinoma T1	0.91	1	0.95	10
Neurocitoma T1	0.95	0.95	0.95	19
Oligodendroglioma T1C+	1	0.91	0.95	11
Neurocitoma T1C+	0.97	0.94	0.96	34
Oligodendroglioma T1	1	0.92	0.96	13
_NORMAL T1	0.97	0.97	0.97	37
Germinoma T1C+	1	1	1	6
Tuberculoma T1	1	1	1	4

For the 214 T1 images accuracy was 0.864. For the 255 T1C+ images accuracy was 0.859. For the 203 T2 images the accuracy was substantially lower at 0.778. One neuroimaging study found lower accuracy for classifying T2 images (4).

Table 6. Accuracy by Contrast Image Types

	n	%
T1	214	86.4
T1C+	255	85.9
T2	203	77.8

Summary and conclusions.

The Resnet-18 network achieves nearly 84% accuracy on the test set after resizing images to 630X630. The highest accuracy is for T1 images at 86.4% and T1C+ images at 85.9%. T2 images had a lower accuracy at 77.8%. Improvements might be possible with generative adversarial networks.

References

1. https://www.kaggle.com/datasets/fernando2rad/brain-tumor-mri-images-44c
2. Kawahara D, Nagata Y. T1-weighted and T2-weighted MRI image synthesis with convolutional generative adversarial networks. Rep Pract Oncol Radiother. 2021 Feb 25;26(1):35-42. doi: 10.5603/RPOR.a2021.0005. PMID: 33948300; PMCID: PMC8086713.
3. https://www.mayoclinic.org/diseases-conditions/meningioma/
4. Alves AFF, Miranda JRA, Reis F, de Souza SAS, Alves LLR, Feitoza LM, de Castro JTS, de Pina DR. Inflammatory lesions and brain tumors: is it possible to differentiate them based on texture features in magnetic resonance imaging? J Venom Anim Toxins Incl Trop Dis. 2020 Sep 4;26:e20200011. doi: 10.1590/1678-9199-JVATITD-2020-0011. PMID: 32952531; PMCID: PMC7473508.
5. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec
6. https://pytorch.org/docs
7. He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

Appendix

```
Resnet18(
  (resnet): 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=44, bias=True)
 )
)
```