CS464 - PROGRESS REPORT

BRAIN TUMOR DETECTOR

GROUP 1

- Ahmet Hakan Yılmaz - 21803399
- Akın Kutlu - 21803504
- Esra Genç - 21901962
- Osman Semih Tiryaki - 21801994
- Zehra Erdem - 21801977

1. INTRODUCTION

Many different techniques have been used to detect brain tumors and humanity struggled to understand the cause and features of such a deadly brain disorder. However, a recent technology called Magnetic Resonance Imaging(MRI) scan produced very accurate results when detecting brain tumors. Today, MRI scan is considered as the best way to detect tumors around the brain and spinal cord area [1]. Instead of being looked at by the doctors, MRI scan results can be analyzed by a machine learning model to produce accurate results. This project aims to train a machine learning model that can classify MRI scan results as normal or tumor detected with the type of the tumor among the 3 types of tumor: Glioma, Meningioma, Pituitary. Thus, the brain tumors will be detectable much precisely and instantly.

In this project, three machine learning methods will be used to train three different models to be able to analyze differences among them. Logistic Regression, even though it is not widely used for image classification tasks, is one of the methods used in this project to analyze the accuracy rate of a model trained with a slightly non-optimal method. However, logistic regression classifies images within a decreased computational time so this part also demonstrates the tradeoff between accuracy and pace of the model. Second model to train will be trained using MLP. MLP is one of the more accurate techniques so the comparison of it with CNN will be demonstrated and interpreted in terms of accuracy, pace and ease of implementation. Lastly, the most commonly used technique in image classification tasks: CNN will be used to train another model. Since CNN models include convolutional layers which use neighboring points to calculate the one point, CNN generally has accurate results in image processing. Thus, the most accurate results are expected from the CNN model.

2. BACKGROUND INFORMATION

2.1. Logistic Regression

Logistic regression is one of supervised machine techniques and tries to maximize likelihood function. Gradient descent algorithms are used to find the optimal coefficients of the model. In our project we used multinomial logistic regression since we have 4 different classes for classification. We used stochastic gradient descent as the optimizer.

2.2. MLP

The most basic type of artificial neural network is called a Multilayer Perceptron (MLP) and it is widely used to remove noise from input feature sets [2]. They convert a sequence of inputs into a single output between 0 and 1 using a set of perceptrons, or equations with inputs, outputs, and weights. Once another layer of perceptrons receives that signal, the process is repeated until only one output remains (or set of outputs, depending on the function of the MLP). The input layer, at least one hidden layer, and the output layer are the minimum number of layers that make up an MLP [3].

2.3. CNN

CNN is used for pattern recognition in the images and so for image classification. Different from MLP, CNN has convolutional layers. These layers have a kernel with specified sizes. Instead of applying different parameters for every feature, in the convolutional layer, every part of the picture that has the same size as the kernel is multiplied by the same kernel. This means they have shared parameters and if a kernel causes the recognition of an image, since the same kernel will be used in the other parts, other instances of this object within the image can be recognized as well. Therefore, CNN is one of the successful models for pattern recognition and classification [4].

3. WHAT IS DONE

3.1. Preprocess

The motivation behind the preprocessing was increasing model performance and accuracy. Two different libraries' preprocessing functions were examined and we decided to use PyTorch's functions [5]. The preprocessing steps: resize, randomHorizontalFlip, toTensor, and Normalize. After resizing, the size of the dataset decreased and became easy to process. The tumor can be in different parts of the brain and the shape can be different. So, using RandomHorizontalFlip was appropriate for distributing the tumor into different parts of the images. Before normalizing the image, the image should be transformed to tensor. toTensor transforms the image to a kind of array. Normalize function has 2 parameters: mean and standard deviation. With tensor functions we found the necessary parameters and normalized the image [6].

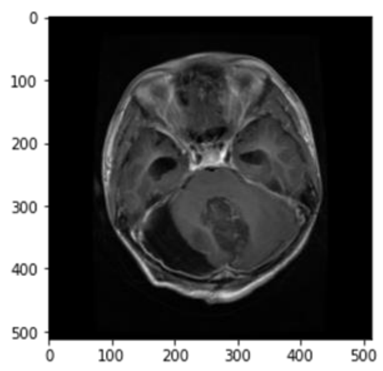
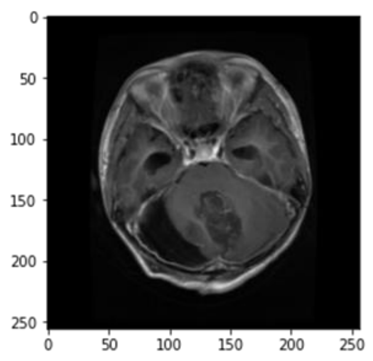
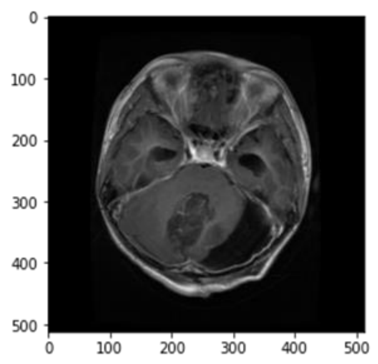
		

Figure 1: Original Image	Figure 2: Resized Image	Figure 3:Horizontal Flipped Image

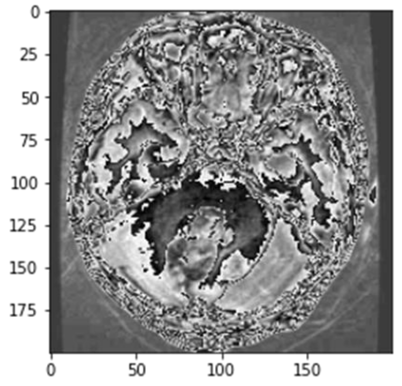
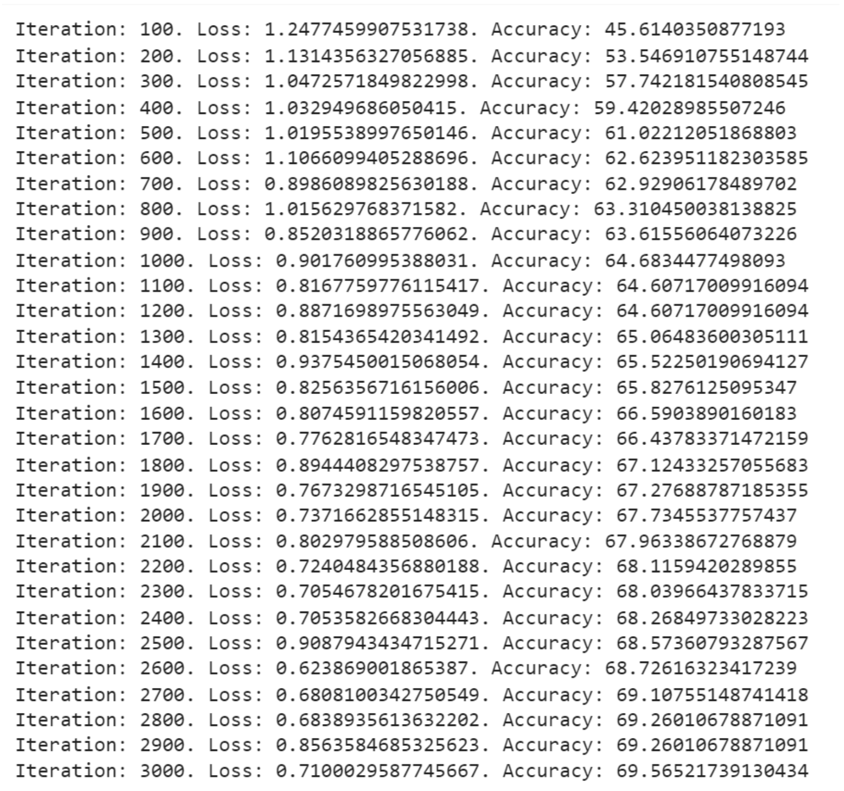


Figure 4:After all operations

3.2. Logistic Regression

We used the `Torch` library for logistic regression. Firstly we selected batch size as 60, and the number of iteration as 3000 over those sets with batch size 60. Therefore number of epochs we apply is equal to 3000*60 / 5712 approximately 31. We created a `train_loader` and `test_loader` object by using the `Dataloader` function of the torch library according to the batch size we decided. Our logistic regression model includes a single linear layer that that gets input as 256*256*3 = 196608 features and 4 output features as we have 4 different brain tumor types for classification. While creating an optimizer for our parameters we used Stochastic gradient descent. We selected `learning_rate` as 0.00005 for our optimizer. We used `CrossEnyropyLoss()` for calculating loss in every iteration. We updated our parameters using the optimizer object. The accuracy rate for our logistic model is approximately 69.5% at the end of 3000 th iteration.

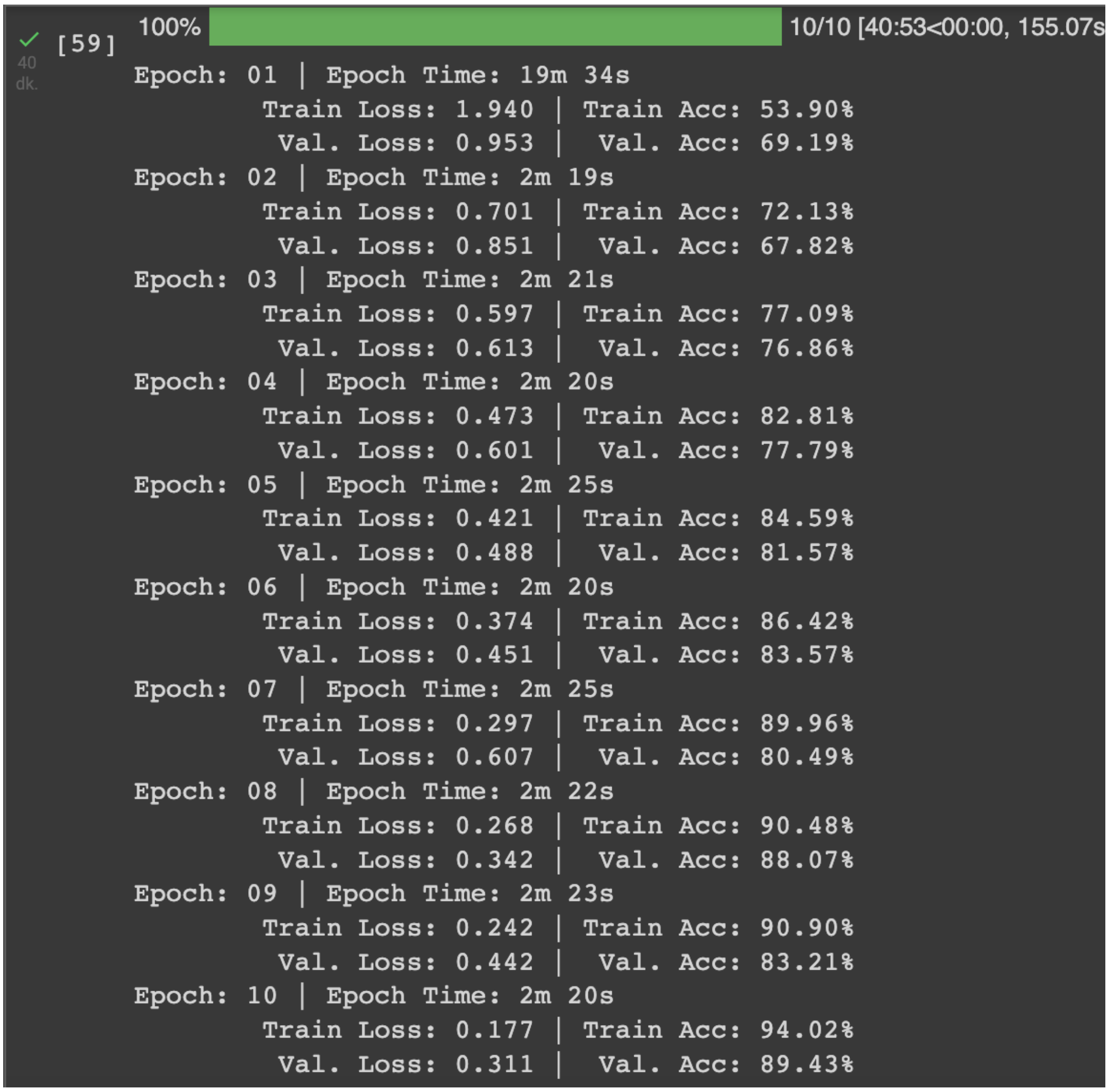


```
Iteration: 100. Loss: 1.2477459907531738. Accuracy: 45.6140350877193
Iteration: 200. Loss: 1.1314356327056885. Accuracy: 53.546910755148744
Iteration: 300. Loss: 1.0472571849822998. Accuracy: 57.742181540808545
Iteration: 400. Loss: 1.032949686050415. Accuracy: 59.42028985507246
Iteration: 500. Loss: 1.0195538997650146. Accuracy: 61.02212051868803
Iteration: 600. Loss: 1.1066099405288696. Accuracy: 62.623951182303585
Iteration: 700. Loss: 0.8986089825630188. Accuracy: 62.92906178489702
Iteration: 800. Loss: 1.015629768371582. Accuracy: 63.310450038138825
Iteration: 900. Loss: 0.8520318865776062. Accuracy: 63.61556064073226
Iteration: 1000. Loss: 0.901760995388031. Accuracy: 64.6834477498093
Iteration: 1100. Loss: 0.8167759776115417. Accuracy: 64.60717009916094
Iteration: 1200. Loss: 0.8871698975563049. Accuracy: 64.60717009916094
Iteration: 1300. Loss: 0.8154365420341492. Accuracy: 65.06483600305111
Iteration: 1400. Loss: 0.9375450015068054. Accuracy: 65.52250190694127
Iteration: 1500. Loss: 0.8256356716156006. Accuracy: 65.8276125095347
Iteration: 1600. Loss: 0.8074591159820557. Accuracy: 66.5903890160183
Iteration: 1700. Loss: 0.7762816548347473. Accuracy: 66.43783371472159
Iteration: 1800. Loss: 0.8944408297538757. Accuracy: 67.12433257055683
Iteration: 1900. Loss: 0.7673298716545105. Accuracy: 67.27688787185355
Iteration: 2000. Loss: 0.7371662855148315. Accuracy: 67.7345537757437
Iteration: 2100. Loss: 0.802979588508606. Accuracy: 67.96338672768879
Iteration: 2200. Loss: 0.7240484356880188. Accuracy: 68.1159420289855
Iteration: 2300. Loss: 0.7054678201675415. Accuracy: 68.03966437833715
Iteration: 2400. Loss: 0.7053582668304443. Accuracy: 68.26849733028223
Iteration: 2500. Loss: 0.9087943434715271. Accuracy: 68.57360793287567
Iteration: 2600. Loss: 0.623869001865387. Accuracy: 68.72616323417239
Iteration: 2700. Loss: 0.6808100342750549. Accuracy: 69.10755148741418
Iteration: 2800. Loss: 0.6838935613632202. Accuracy: 69.26010678871091
Iteration: 2900. Loss: 0.8563584685325623. Accuracy: 69.26010678871091
Iteration: 3000. Loss: 0.7100029587745667. Accuracy: 69.56521739130434
```

Figure 5: Results of the Logistic Regression Model

3.3. MLP

We have implemented an MLP model using the `Torch` library. First of all, we preprocessed our data and then divided it to train and test data. We selected a batch size equal to 100, and we set the number of epochs as 10. We created a `train_loader` and `test_loader` object by using the `torch.utils.data.DataLoader` function according to the batch size we decided. Our MLP model is formed from three layers which include an input layer that that gets input in input dimension 256*256*3 = 196608 and reduces it to 250 dimensions, one hidden layer that reduces the dimension to 100 and an output layer which gives the output in 4 dimensions as we have 4 different brain tumor types for classification. Inside the MLP model we used `torch.nn.functional.relu` method as it applies the rectified linear unit function element-wise. While creating an optimizer for our parameters we used `torch.optim.Adam()`. We used `torch.nn.CrossEnyropyLoss()` for calculating loss in every iteration. The accuracy rate for the MLP model is approximately 89.43% at the end of the training and validation process.

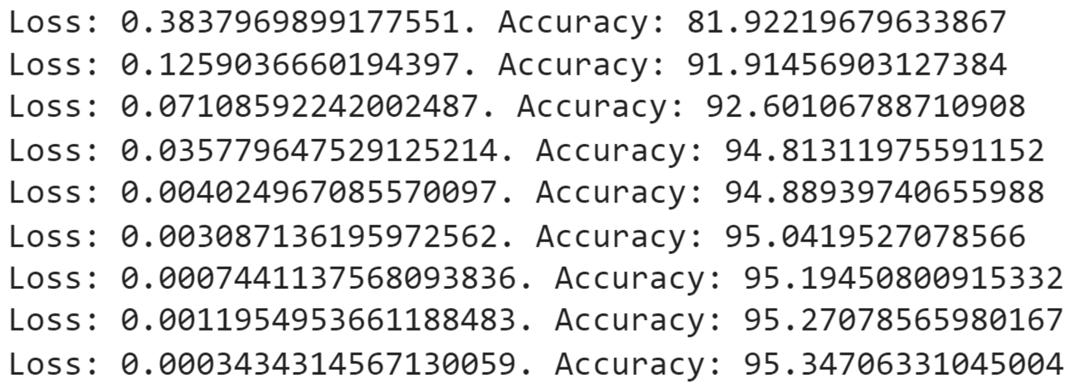


```
✓ [59]   100% 10/10 [40:53<00:00, 155.07s
40
dk.
         Epoch: 01 | Epoch Time: 19m 34s
                 Train Loss: 1.940 | Train Acc: 53.90%
                  Val. Loss: 0.953 |  Val. Acc: 69.19%
         Epoch: 02 | Epoch Time: 2m 19s
                 Train Loss: 0.701 | Train Acc: 72.13%
                  Val. Loss: 0.851 |  Val. Acc: 67.82%
         Epoch: 03 | Epoch Time: 2m 21s
                 Train Loss: 0.597 | Train Acc: 77.09%
                  Val. Loss: 0.613 |  Val. Acc: 76.86%
         Epoch: 04 | Epoch Time: 2m 20s
                 Train Loss: 0.473 | Train Acc: 82.81%
                  Val. Loss: 0.601 |  Val. Acc: 77.79%
         Epoch: 05 | Epoch Time: 2m 25s
                 Train Loss: 0.421 | Train Acc: 84.59%
                  Val. Loss: 0.488 |  Val. Acc: 81.57%
         Epoch: 06 | Epoch Time: 2m 20s
                 Train Loss: 0.374 | Train Acc: 86.42%
                  Val. Loss: 0.451 |  Val. Acc: 83.57%
         Epoch: 07 | Epoch Time: 2m 25s
                 Train Loss: 0.297 | Train Acc: 89.96%
                  Val. Loss: 0.607 |  Val. Acc: 80.49%
         Epoch: 08 | Epoch Time: 2m 22s
                 Train Loss: 0.268 | Train Acc: 90.48%
                  Val. Loss: 0.342 |  Val. Acc: 88.07%
         Epoch: 09 | Epoch Time: 2m 23s
                 Train Loss: 0.242 | Train Acc: 90.90%
                  Val. Loss: 0.442 |  Val. Acc: 83.21%
         Epoch: 10 | Epoch Time: 2m 20s
                 Train Loss: 0.177 | Train Acc: 94.02%
                  Val. Loss: 0.311 |  Val. Acc: 89.43%
```

Figure 6: Results of the MLP Model

3.4. CNN

The `Torch` library is used for the implementation of CNN. After preprocessing the data, the `train_loader` and `test_loader` with batch size 100 are created by using the `DataLoader` of Torch. Number of iterations is chosen as 1000 so the number of epochs becomes 1000*100/5712 = 17. In the model two convolutional layers with the kernel size (5,5) is used. After every convolutional layer ReLu activation function and max pooling with size (2,2) are applied. After that, the feature map is flattened and two linear layers are applied so that the last size becomes four. `CrossEnyropyLoss` is used as the loss function and `Adam` is used as the optimizer during training. The validation accuracy after the last iteration improved to 95.35%.



```
Loss: 0.3837969899177551. Accuracy: 81.92219679633867
Loss: 0.1259036660194397. Accuracy: 91.91456903127384
Loss: 0.07108592242002487. Accuracy: 92.60106788710908
Loss: 0.035779647529125214. Accuracy: 94.81311975591152
Loss: 0.004024967085570097. Accuracy: 94.88939740655988
Loss: 0.003087136195972562. Accuracy: 95.0419527078566
Loss: 0.0007441137568093836. Accuracy: 95.19450800915332
Loss: 0.0011954953661188483. Accuracy: 95.27078565980167
Loss: 0.0003434314567130059. Accuracy: 95.34706331045004
```

Figure 7: Results of the CNN Model

4. WHAT REMAINS TO BE DONE

- The implementation of MLP and CNN will be finalized, the final version of convolutional and fully connected layers and their parameters will be decided.
- The Finalized MLP and CNN models will be fed with test data and results will be obtained for interpretation. Confusion matrices for all three models will be obtained and presented.
- If spare time is left, transfer learning and random forest regression can be used to see differences.

5. DIVISION OF WORK AMONG TEAM MEMBERS

- Preprocess → Akın Kutlu, Esra Genç, Osman Semih Tiryaki
- Logistic Regression → Ahmet Hakan Yılmaz, Zehra Erdem
- MLP → Akın Kutlu, Esra Genç
- CNN → Ahmet Hakan Yılmaz, Osman Semih Tiryaki, Zehra Erdem

6. REFERENCES

[1] “Tests for brain and spinal cord tumors in adults,” *American Cancer Society*. [Online]. Available: https://www.cancer.org/cancer/brain-spinal-cord-tumors-adults/detection-diagnosis-staging/how-diagnosed.html. [Accessed: 03-Dec-2022].

[2] G. Latif, M. Mohsin Butt, A. H. Khan, M. Omair Butt, and J. F. Al-Asad, “Automatic Multimodal Brain Image Classification using MLP and 3D glioma tumor reconstruction,” *2017 9th IEEE-GCC Conference and Exhibition (GCCCE)*, 2017.

[3] T. Prole, “What is an MLP, and why should you care?,” *Medium*, 26-Apr-2022. [Online]. Available: https://towardsdatascience.com/what-is-an-mlp-and-why-should-you-care-bfa06741a183. [Accessed: 02-Dec-2022].

[4] K. O'Shea and R. Nash, “An introduction to Convolutional Neural Networks,” *arXiv.org*, 02-Dec-2015. [Online]. Available: https://arxiv.org/abs/1511.08458. [Accessed: 03-Dec-2022].

[5] S. V. R, “Pytorch transformations: 10 pytorch transformations for Data scientists,” *Analytics Vidhya*, 22-Apr-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/10-pytorch-transformations-you-need-to-know/. [Accessed: 03-Dec-2022].

[6]“Transforming and augmenting images,” *Transforming and augmenting images - Torchvision 0.14 documentation*. [Online]. Available: https://pytorch.org/vision/stable/transforms.html. [Accessed: 03-Dec-2022].