A Transfer Learning Approach to Detecting Brain Tumors

Sam Felsted (felsteds@oregonstate.edu)
ENGR100 Automating the Future, Oregon State University, Corvallis, OR 97331, USA

Abstract — One of the first steps to curing cancer is detection of tumors. Recent AI advances have proven to be very effective in image classification problems such as this. This paper explores several CNN approaches to detecting brain tumors in MRI scans. A final model includes cross training the VGG 16 image classifier to learn tumor signatures and was able to reach a validation accuracy of 88%.

Index Terms — Convolutional Neural Networks, VGG 16, Keras, MRI scans, Cancer, Glioma, Meningioma, Pituitary

I. INTRODUCTION

Traditionally, MRI scans have been reviewed by health professionals to determine if there is a tumor and type. AI tools may be used to aid in reviewing scans and speeding up the diagnosis process to allow for doctors to spend less time detecting and more time treating these diseases[1]. This study uses a convolutional neural network to speed up the detection process.

While the ideas behind convolutional neural networks have been around since at least the 1980s, it is only recently that they have begun to take off. This is because of several factors such as the increase in availability of large, labeled datasets and increases in computer hardware power that let deep learning become more viable [2].

In traditional programming, the programmer constructs a set of rules or instructions for the computer. Programs are written to define what happens when given specific input and how to respond accordingly. However, in machine learning, the programs give the computer the input and desired output of a function. The computer will then attempt to generate a function that optimizes the accuracy of the model in a process called fitting [3, p. 4].

A deep learning model is defined by its layers. The layers of a network are mini transformation functions of data that chain together to ultimately output a prediction of the data. Layers are defined by the programmer before fitting begins and are parameterized by their weights. To learn a dataset, a deep learning model passes the data through each layer of the network in a feedforward manner until it outputs a prediction for the input. The outputs of the model are compared to the labels of the dataset and then the parameters are adjusted in a manner to increase accuracy. [3, p. 9].

Finding the optimal weights for the network to maximize accuracy is a difficult task as there can easily be hundreds of thousands of tunable weights that each impact the accuracy of the model. A common optimization algorithm for such models is called a "gradient descent" [4]. To better understand how gradient descent works, think back to a differential calculus class. Optimizing single variable functions is often one of the first applications of differentiation taught.

Take some arbitrary cost function and differentiate it to obtain local minimum or maximums and calculate the lowest possible cost. In gradient descent, you take the partial derivative of the model's cost (defined as how far the output is from what you expected [3, p. 9]) and adjust the weights in the direction that minimizes the cost of the network. When the derivative is steeper, the model will adjust the parameters faster. The steepness will decrease until the model converges at a minimum cost. [4].

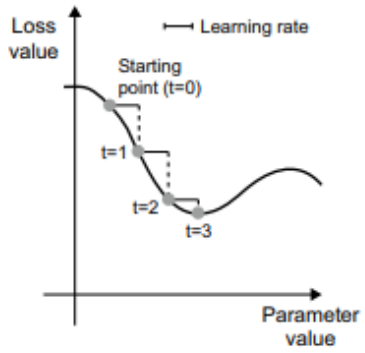

Fig. 1. This is an excellent visualization of minimizing loss of the cost function in one dimension. Note how the function settles on a local minimum and may not be a true minimum of the function. [3, p. 53]

Convolutional neural networks are a subset of deep learning models that are categorized by their "sick" convolutional layers. The convolution operation involves taking subsections of the data input and extracting features from it. These features are then learned in a traditional neural network style. Think about it as a sliding magnifying glass over the image extracting pieces. An example would be an animal classifier may have convolutional layers that detect edges. Another convolutional layer learns what a "leg" and a "tail" are, and the traditional layers may learn that 4 legs and a tail can be classified as a dog [6]. Reference 5 is an amazing supplement video to understanding convolutional operation and **Fig 2** illustrates convolutional feature detection.

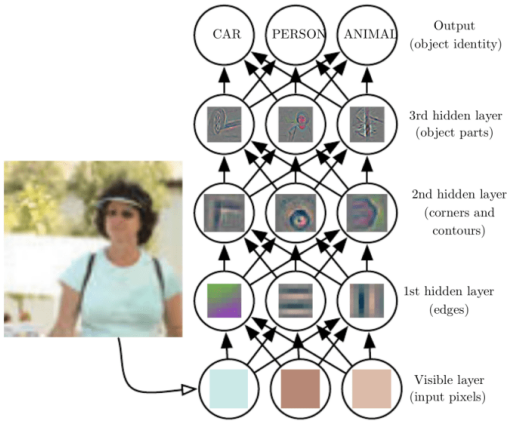

Fig. 2. In this example, an image of a women is inputted into the CNN. Features are extracted from the convolutional layers and higher order features such as her arm are identified [6].

With this preliminary knowledge of CNNs, we can now dive into this paper. Section II covers information about the dataset used. Section III goes over a couple of model architecture tests and their performances. Section IV is a conclusion.

II. DATA. DATA. DATA!

The most important thing about AI is garbage in, garbage out. Bad data will give you bad results. For this model, data has been pulled from Kaggle [7]. The dataset contains 4 directories of JPG of size 256 by 256. images with the name of the directory corresponding to the label of the directory. An example of the data is illustrated in **Fig 3**.

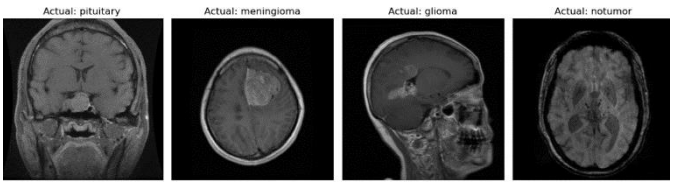

Fig. 3. Sample data from the datasets. Tumors can be seen in the first 3 images.

An important thing to always note is the meta data of the dataset. The source of the data has not been specified and the dataset. The Kaggle author has claimed that the set has been synthetically developed for the purpose of CNN research and the dataset should not be used for clinical diagnosis. It was posted publicly in September of 2023.

In this dataset, there are 1311 files (300 Pituitary, 306 Meningioma, 300 Glioma, 405 no tumor) [6]. They have been randomly shuffled and 1049 are used for training and 262 have been used for testing. (80/20 split).

III. CNN ARCHITECTURE, TRAINING, AND VALIDATION

A couple of model architectures were tried in this study. One of the first models I designed and trained on was based on VGG16. Architecture is pictured in **figure 4**. VGG16 was selected due to its robustness and my own personal familiarity with the architecture.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 250, 250, 16)      2368

 max_pooling2d_3 (MaxPoolin  (None, 125, 125, 16)      0
 g2D)

 conv2d_7 (Conv2D)           (None, 123, 123, 16)      2320

 conv2d_8 (Conv2D)           (None, 121, 121, 16)      2320

 max_pooling2d_4 (MaxPoolin  (None, 60, 60, 16)        0
 g2D)

 conv2d_9 (Conv2D)           (None, 58, 58, 32)        4640

 conv2d_10 (Conv2D)          (None, 56, 56, 32)        9248

 max_pooling2d_5 (MaxPoolin  (None, 28, 28, 32)        0
 g2D)

 flatten_2 (Flatten)         (None, 25088)             0

 dense_4 (Dense)             (None, 1024)              25691136

 dense_5 (Dense)             (None, 128)               131200

 dense_6 (Dense)             (None, 4)                 516

=================================================================
Total params: 25843748 (98.59 MB)
Trainable params: 25843748 (98.59 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 4. Model architecture of my custom CNN based on a VGG architecture.

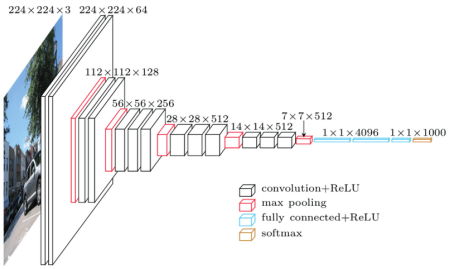

Fig. 5. Supplementary diagram of the typical architecture of VGG16 [8]. My own classifier differed as described **Fig 4**.

This model, however, did not prove to be as effective as I would've hoped. The model quickly began overfitting (memorizing the learning set but not actually performing better on data it hasn't seen before). around 6 epochs **(fig. 6)**. My theory is this is because there are not enough convolutional layers to fully detect all features of the MRI scans. However, due to limitations in my own personal computer hardware, I was not able to efficiently train a custom larger model. Due to the custom nature of the CNN, I was able to pull a GradCam of the model to see the features that it learned **(fig. 7).**

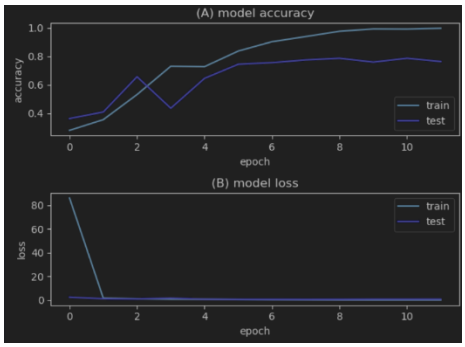

Fig. 6. Learning history of the first CNN. Training performance and testing performance really start to diverge at 6 epochs.

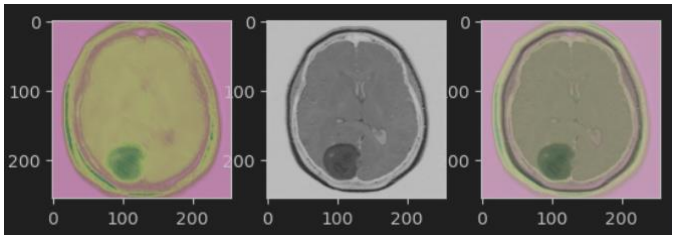

Fig. 7. GradCam of the first CNN. You can see the learned features of the First CNN. The section of the MRI with the "sus" spot is lit up in the activation of the last convolutional layer.

Because of my computing limitations, I decided to transfer train VGG16 itself. Transfer learning involves taking a pretrained model and retraining the final layers for new class detection. The already trained convolutional layers will feed into a new network thus allowing for a computationally efficient approach to classification.

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 512)               14714688

 flatten_6 (Flatten)         (None, 512)               0

 dense_14 (Dense)            (None, 1024)              525312

 dense_15 (Dense)            (None, 4)                 4100

=================================================================
Total params: 15244100 (58.15 MB)
Trainable params: 529412 (2.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

Fig. 8. Model architecture of transfer learning VGG16

This was much more effective, leading to a testing accuracy of 88%. Overfitting is still a problem. Future approaches may involve retraining the last couple convolutional layers of VGG16 for more accurate feature detection.

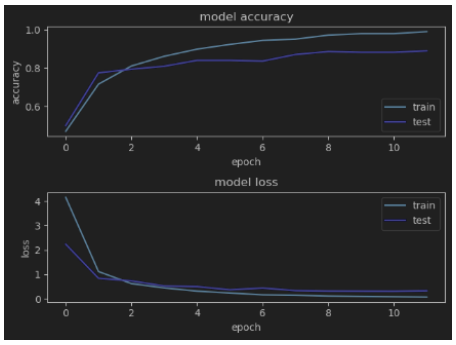

Fig. 9. Learning history of the second CNN

IV. Conclusion

As faster computer hardware is released and as new methods arise for deep learning, we will likely see models that get up to 99.9% accurate. AI is revolutionizing the world. While having an AI that completely replaces health care professionals is likely very far away, we may soon see AI companions assisting doctors in various places.

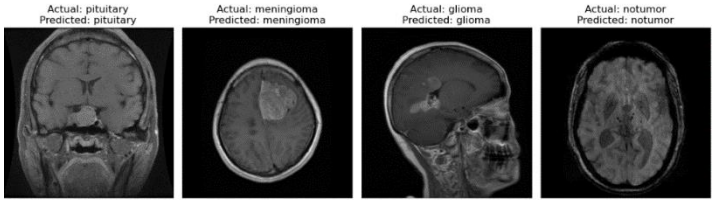

Fig. 10. Image from before with classifications from VGG16

REFERENCES

[1] "Brain tumor - Diagnosis and treatment - Mayo Clinic," www.mayoclinic.org. https://www.mayoclinic.org/diseases-conditions/brain-tumor/diagnosis-treatment/drc-20350088

[2] R. Demush, "A brief history of computer vision (and convolutional neural networks)," HackerNoon, https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3 (accessed Dec. 14, 2023).

[3] Francois Chollet, Deep Learning with Python, Second Edition, Manning, 2021. https://ieeexplore.ieee.org/document/10280481 (accessed Dec. 14, 2023).

[4] "What is gradient descent?," IBM, https://www.ibm.com/topics/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates (accessed Dec. 14, 2023).

[5] Sanderson, Grant. "But What Is a Convolution?" www.youtube.com, 18 Nov. 2018, www.youtube.com/watch?v=KuXjwB4LzSA.

[6] M. Mandal, "Introduction to convolutional neural networks (CNN)," Analytics Vidhya, https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ (accessed Dec. 14, 2023).

[7] S. Gupta, "Brain MRI scans for Brain Tumor Classification," Kaggle, https://www.kaggle.com/datasets/shreyag1103/brain-mri-scans-for-brain-tumor-classification/data (accessed Dec. 14, 2023).

[8] "Understanding VGG16: Concepts, architecture, and performance," Datagen, https://datagen.tech/guides/computer-vision/vgg16 / (accessed Dec. 14, 2023).

[9] K. Team, "Keras Documentation: Grad-cam class activation visualization," Keras, https://keras.io/examples/vision/grad_cam/ (accessed Dec. 14, 2023).