U-Net Network for MRI Brain-Tumor Segmentation Using the BraTS 2020 Dataset

Applied Machine Learning Project

Gabriel Guerra Trigo ggt2112@columbia.edu
Devika Gumaste dg3370@columbia.edu
Haiwen Kou hk3283@columbia.edu
Thomas Lim tl2977@columbia.edu
Yuexin Zhu yz4725@columbia.edu

December 2023

1 Task Description

We begin by presenting a formal description of the task at hand: we intend to train a model to perform image segmentation of brain MRI images, classifying each pixel as being part (or not) of different parts of a tumor. The dataset consists of 57195 image slices from 369 volumes. Each image has a resolution of 240×240 pixels with 4 channels, totalling dimensions $(244, 244, 4)$. The channel corresponds to the Native (T1), post-contrast T1-weighted (T1Gd), T2-weighted (T2), and T2 Fluid Attenuated Inversion Recovery (T2-FLAIR) variants. Each slice is accompanied by an expert segmentation mask, which classifies each pixel as one of four labels: Normal(label 0), Necrotic and Non-Enhancing Tumor Core (label 1), Peritumoral Edema (label 2), and GD-enhancing tumor (label 3).

2 Methodology

The U-Net architecture we intend to use is comprised of three main parts: an encoder, a decoder and the skip connections. The encoder portion features several convolutional layers followed by max-pooling layers, which are responsible for extracting relevant features from the images. For computer vision tasks such as this, it is important to highlight that the encoder does not need to be task-specific. This means that for our task, we can choose between training an encoder from scratch, or leveraging pre-trained encoders that have been developed with more data and resources than what we have available.

The decoder on the other hand is responsible for reconstructing a high resolution output, and for that reason it evidently must be trained specific to the task. Finally, the skip connections take the outputs from downsampling layers (for example, the first downsampling layer is paired with the last upsampling layer) and concatenates them to the inputs to upsampling layers. This combines low-level features with high-level ones and allows the model to grasp the complete picture. With these concepts in mind, we opted to train two models, the first using a pre-trained encoder and the second training everything from scratch. This allowed us to compare the quality of both models, both through the lens of prediction quality and time/resources necessary to train. Both models were trained using GTX 1660 Super graphics card with CUDA. Both models were trained with a batch size of 20 and the Adam optimizer.

3 Model(s) Description

For the **pre-trained encoder model**, we made use of MobileNetV2 as the encoder, setting **trainable = false** to prevent further training due model size and complexity. For upsampling blocks, we used the implementation from pix2pix. Since the model was trained with 3 input channels, we arbitrarily discarded the last channel (T2-FLAIR), since any of the 4 channels could be discarded. In a more in depth analysis, one could try discarding each channel and seeing what yielded the best results, but we did not perform this analysis. To suit the pre-trained model, we reescaled the input and mask resolution to be 224×224 pixels (see data preprocessing section), as this is the maximum input size supported by the model. We used 4 downsampling blocks and 4 upsampling blocks, such that the input dimesions followed the sequence $[224, 112, 56, 28, 14]$ during downsampling, and the inverse during upsampling.

For the **trained from scratch model**, we were not restrained by the limitations of the pre-trained encoder, so we kept all 4 input channels and kept the input at its original resolution of 240×240. For the downsampling portion, we used blocks of Conv2D layers followed by MaxPool2D layers. The upsampling portion was similar, but The MaxPool2D layers were replaced by UpSampling2D layers. Similarly to the pre-trained encoder model, we used 4 downsampling blocks and 4 upsampling blocks, and the input dimensions followed the sequence $[240, 120, 60, 30, 15]$.

4 Data Preprocessing and Imbalance

4.1 Data Splitting

Due to the nature of the dataset, where several images correspond to the same volume, we opted to perform a structured split based on each image's volume. This means that all images from the same volume were on the same split, while the volumes themselves were split randomly. We used slkearn's GroupShuffleSplit to do this. This prevents data leakage and ensures true performance when validating and testing.

4.2 Imbalance

The labels in the dataset at hand are extremely unbalanced. Precisely, the label distribution is $\{0 : 98.88\%, 1 : 0.27\%, 2 : 0.58\%, 3 : 0.27\%\}$. Notably, the 0 class (normal) class is much more abundant than the remaining classes. To deal with this, we opted to apply class weights to our model. This was done by setting weights in order to make each class' relevance balanced, as described in scikit-learn.

4.3 Image Reescaling

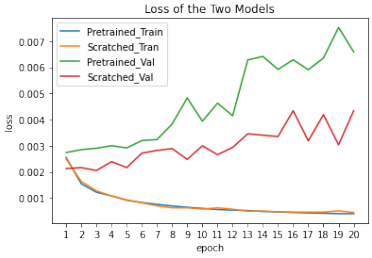
As mentioned section 3.1, we reescaled the images prior to feeding them to the pre-trained encoder to adjust to its accepted dimensions. To do this, we made use of Tensorflow's resize function. To preserve the discrete nature of the labels, we used the "Nearest Neighbor"

method when resizing the image, while the images themselves were treated with the default parameters (bilinear interpolation).

5 Results

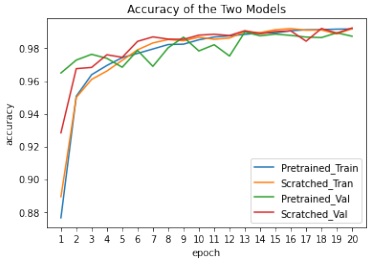
5.1 Loss

Comparing the pre-trained encoder (0.000404) and trained-from-scratch models (0.000450), we observed that the training loss was similar, but the trained-from-scratch model had less validation loss (0.004343 compared to 0.006585).



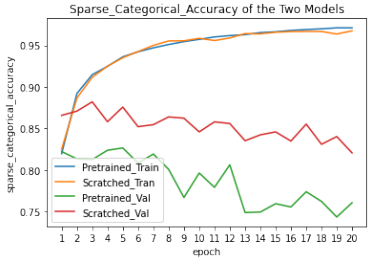
5.2 Accuracy

The training accuracy of the two models were similar, as well as for the validation accuracy, although the trained-from-scratch model had a slightly higher validation accuracy (0.992478 compared to 0.987424). This may be explained because all 4 input channels were preserved, and the input was kept at the original resolution, improving the accuracy.



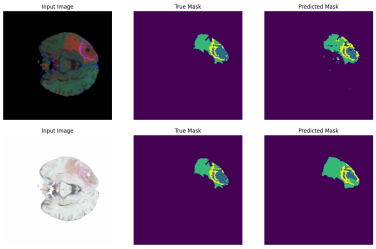
5.3 Sparse Categorical Accuracy

To account for label imbalance, we evaluated Sparse Categorical Accuracy. The trained-from-scratch model had a much higher score (0.820632 compared to 0.760608).



5.4 Sample Output

This figure shows a sample output from both models. The first row is the result from the pre-trained model, and the second is from the trained-from-scratch model.



5.5 Conclusion

The trained-from-scratch (TFS) model shows superior performance over the pre-trained encoder, evidenced by better loss metrics, accuracy, and sparse categorical accuracy. The TFS model's loss indicates stronger generalization on unseen data, while the sparse categorical accuracy suggests it copes better with class imbalances, typical of real-world data. This highlights the TFS model's clear advantages in performance. For future steps, we could employ regularization or other techniques to mitigate overfitting issues.

References

[1] "TensorFlow U-Net Example." [https://www.bing.com/search?q=tensorflow+unet+example].

[2] C. Versloot, "How to Build a U-Net for Image Segmentation with TensorFlow and Keras," 2023. [https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-build-a-u-net-for-image-segmentation-with-tensorflow-and-keras.md].

[3] [https://www.kaggle.com/datasets/awsaf49/brats2020-training-data/data].

[4] B. H. Menze et al., "The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)," in IEEE Transactions on Medical Imaging, vol. 34, no. 10, pp. 1993-2024, 2015. DOI: 10.1109/TMI.2014.2377694.

[5] S. Bakas et al., "Advancing The Cancer Genome Atlas glioma MRI collections with expert segmentation labels and radiomic features," in Nature Scientific Data, vol. 4, article 170117, 2017. DOI: 10.1038/sdata.2017.117.

[6] S. Bakas et al., "Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge," arXiv preprint arXiv:1811.02629, 2018.