To:	Professor Sabuncu
From:	[Priya Kattappurath (psk92), Kristen Ong (kvo3), Daniel Stabile (dis52)]
Date:	12/11/2020
RE:	Machine Learning Final Project Report

The purpose of this report is to describe the problem, data, and results of our project. We will also offer a visual exploration of the data and show results we obtained.

PROBLEM

Low-grade gliomas (LGGs) are brain tumors that originate from glial cells, and are considered the slowest growing type of glioma in adults to date.[1] These tumor cells are capable of growing undetected into the surrounding brain tissue, with the potential of developing into a more aggressive grade tumor, which can disrupt connections and pressure levels between normal brain cells.[2] In general, LGGs are typically only initially identified through an MRI scan of the brain.[1] These scans offer a detailed view of the tumor and can sometimes provide valuable information for diagnostic purposes.

However, MRI diagnosis and annotations by trained radiologists are time intensive and training a model with deep learning could lead to diagnosis much quicker. As such, there is interest for these MRI scans of low-grade gliomas to be assessed through use of a convolutional neural network for segmentation. In this way, tumor segmentations can be algorithmically-produced, decreasing the time needed for a diagnosis and detecting tumours that may normally go undetected by an average human annotator.

DATA

The dataset we will be using consists of brain MR images with FLAIR abnormality segmentation masks. FLAIR (fluid-attenuated inversion recovery) is a type of MRI sequence used to selectively null the signal for certain tissues.[3] In this case, it removes signals from the cerebrospinal fluid in the images, which allows tumors to be more visible. The images themselves are from The Cancer Imaging Archive, and correspond to 110 patients. These patients are included in The Cancer Genome Atlas LGG collection, where there is data available for FLAIR and genomic cluster data.

The data itself consists of 7858 .tif files (between 40 and 146 per patient). Most cases (101) have 3 sequences available (pre-contrast, FLAIR, and post-contrast). For the cases that have either post- or pre- contrast missing, sequences are replaced with FLAIR sequences, such that all images are 3-channel (RGB). The mask images are 1-channel (black and white). These segmentation masks were approved by a board-certified radiologist at Duke University.[4]

In addition to the .tif images, there is a .csv file, which includes Tumor genomic clusters and patient data.

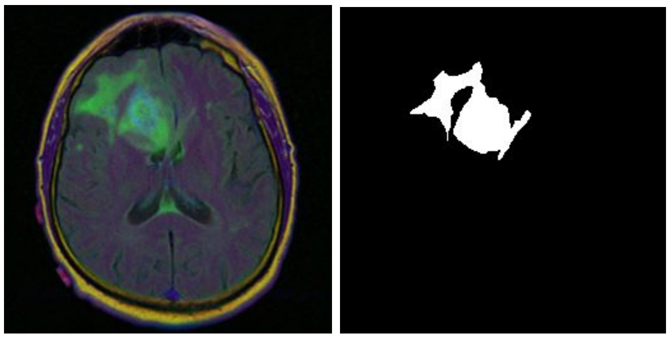

Figure 1. An MRI image of a patient's skull on the left with a mask of an expertly annotated segmentation of the cancer on the right. The image on the left will be the input to our model and the mask will serve as ground truth during training.

CODE

In this section we will provide descriptions of our programming setup and current architecture. Link to Code on Google Colab: https://colab.research.google.com/drive/1Q7IudClorJ97dHyWiLb1KcldYxuJ-jxW?usp=sharing

Google Colaboratory and PyTorch

This convolutional neural network problem, like many other deep learning tasks, requires significant processing power for training. GPUs are used for deep learning tasks, since they have a large number of cores and can therefore do multiple parallel computations faster than CPUs. Because of this, we set up our project in Google Colaboratory (Colab), which allows us to make use of Google's free NVIDIA Tesla K80 GPU. Colab also allows for easy sharing of code between team members through integration between Google Drive and GitHub.

In addition, we utilize PyTorch, a python library similar to Numpy but optimized for use with GPUs. It is often used for deep learning tasks since it is faster and more user-friendly than other options, such as TensorFlow. Furthermore, we were able to find many references that used PyTorch, which are helpful as we process and train our dataset.

Preprocessing

Before training, we perform minimal processing to our data. First, MRI stacks are cropped in each dimension to remove black space. After cropping, the resulting stacks are padded to make them square in the x and y direction and then scaled to 128x128. Dimensions of 128x128 were selected for our images as larger sizes caused our Google Colab notebook to crash due to insufficient RAM. Finally, we normalize our dataset by rescaling all intensities to be between the 10th and 99th intensity percentiles, subtracting the mean of the rescaled dataset, and dividing by its standard deviation.

We implemented skull stripping using a pre-trained model[12] but found that this model not only stripped the skull but often the entire tumour we were trying to segment leading to worse performance.

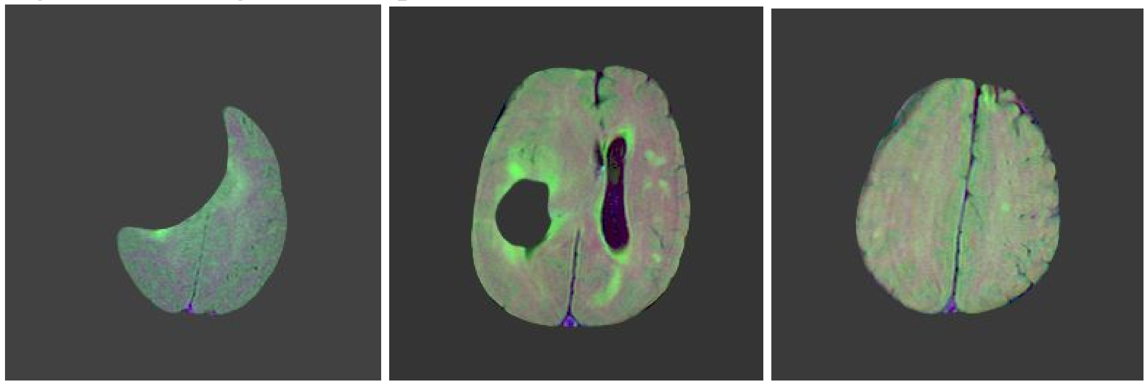
Figure 2: Example images of skull stripping removing the skull and cancer regions. The first two examples demonstrate the skull stipping algorithm removing cancerous sections of the brain.

Data Augmentation

To give the model robustness and the ability to generalize, we implemented data augmentation in 5 different forms. Starting off with some commonly used methods, the MR images were scaled by a randomly selected value between 0 and 5% in either the positive or negative direction. Next, rotation was implemented, which similarly rotated the images by a randomly-selected value in the range of -15 and +15 degrees. A horizontal flipping method was also added, that either flipped or did not flip our images based on a randomly-generated number between 0 and 1 and a probability threshold of 0.5. Then, because it was found in one study that brightness and elastic deformation are the two best augmentation techniques for training a standard 3D U-Net, as they significantly improve Dice loss, brightness was further adjusted and elastic deformation was applied to the training data.[5] Image brightness was adjusted using gamma and gain values selected between 0.8 and 1.2, while elastic deformation was applied using a square deformation grid over the axis [(0, 1), (0, 1)] with displacements sampled from a normal distribution (standard deviation = 2) and smoothing performed by a spline filter. The values used in these methods were selected in correspondence to the methods used in successful studies. Using these augmentation methods generated a larger dataset for the model to train on.

UNet Architecture

The U-Net architecture is the standard for image segmentation. While the original U-Net takes in an input image size of 572x572x3, our input image size is 128x128x3, as chosen in the pre-processing step. Due to this difference, intermediate dimensions in our solution differ from the original U-Net, although the steps are the same. This architecture is broken up into 3 sections: contraction, bottleneck, and expansion. In contraction, two 3x3 convolution + ReLu layers are applied to the input and followed with a 2x2 max pooling with stride 2 for downsampling.[6] The bottleneck layer then applies two 3x3 CNN layers as well as a 2x2 up convolution layer.[6] Finally, the input is then passed on to the expansion section, which upsamples the feature map, applies a 2x2 convolution, concatenates with the corresponding feature map from contraction, and applies two 3x3 convolutions, each of which is followed by a ReLU.[6] With 4 blocks of contractions and 4 blocks of expansions, this totals to 23 convolutional layers in the entire neural network. We experimented with reducing and increasing the number of blocks in the contraction and expansion layers to see the effect on the accuracy of our model. However, we found that the 4 block version was the most successful.

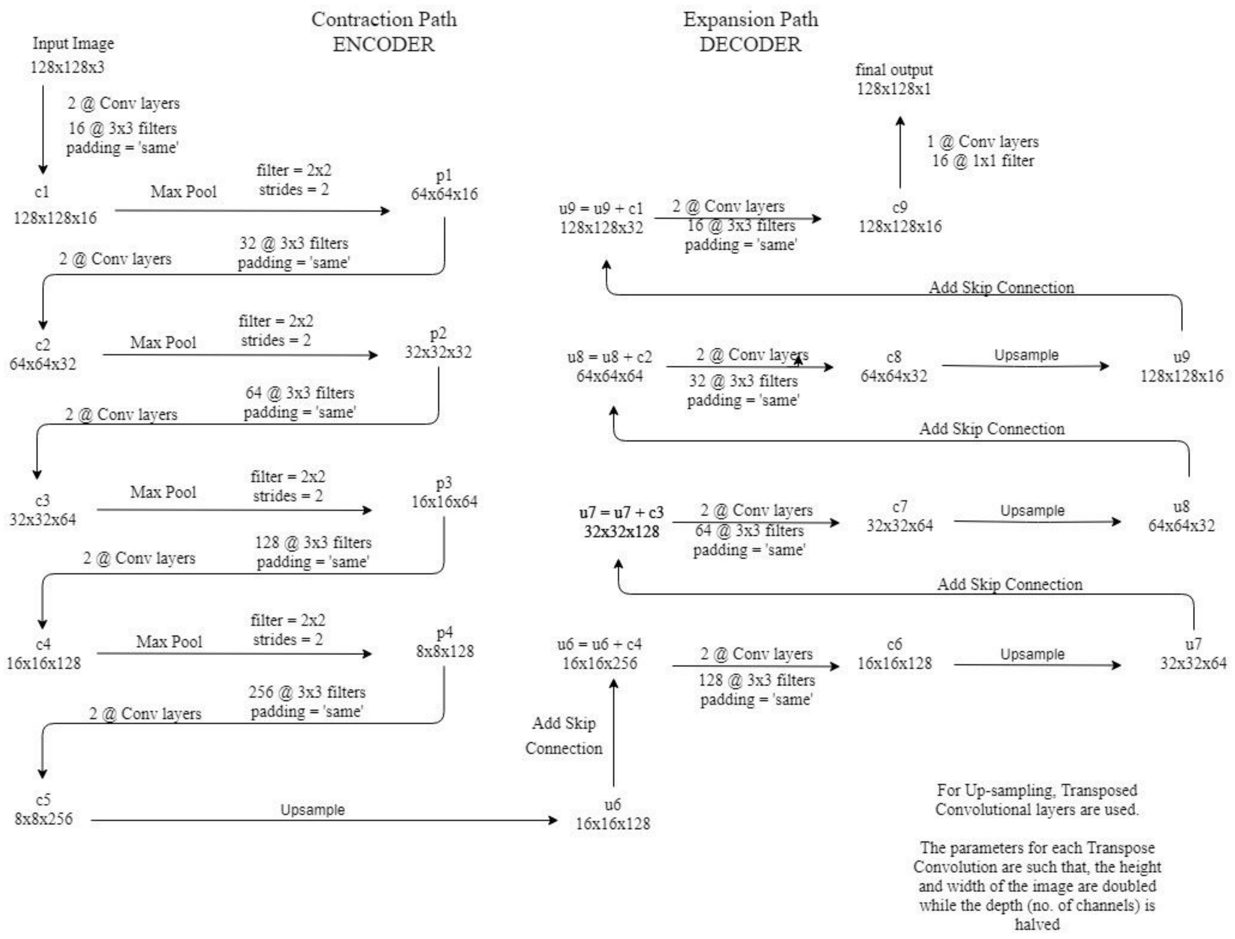

Figure 3: U-Net with input size of 128x128x3.[11]

The loss weighting scheme used by the U-Net architecture to train the network ensures that there is a higher weight at the border of segmented objects for every pixel.[6] This is achieved by applying a pixel-wise softmax on the resultant image, followed by a dice loss function, chosen as it is a popular option for image segmentation tasks. The loss function measures overlap between two samples and is based on the Dice coefficient. This measure ranges from 0 to 1 where a Dice coefficient of 1 denotes perfect and complete overlap.[7] We experimented with two optimizers-- stochastic gradient descent, and Adam, both from PyTorch.

The stochastic gradient descent optimizer is typically used with hyperparameters selected in accordance to those used in the U-Net paper and commonly used for CNNs, which were learning rate = 0.01, momentum = 0.99, batch size = 16, epochs = 50.[8] The parameters used in the PyTorch Adam optimizer are learning rate = 0.001, beta1 = 0.9, betal 2=0.999, and epsilon = 10E-8.[10] We found that the Adam optimizer yielded better validation performance than stochastic gradient descent. We suspect that this is because, while stochastic gradient descent maintains a constant learning rate during training, Adam has an adaptive learning rate that changes based on the average and the uncentered variance of the weights.[10] The beta parameters are responsible for controlling the decay rates of these moving averages, and the default values being set close to 1 result in bias correction towards zero.[10] This is a feature that is unique to Adam and allows it to give quick and accurate results.

RESULTS

For our baseline U-Net, we split our dataset into 90% training and 10% validation. After 50 epochs of training, it can be seen that the dice loss function decreased for both the training and validation datasets, with validation loss dropping from a value of 0.58 to 0.35, and the training loss dropping from 0.46 to 0.04 (Figure 5). The average dice coefficient accuracy, in contrast, increased and leveled off to about 0.81 with the best validation mean accuracy of 0.8314 (Figure 5).

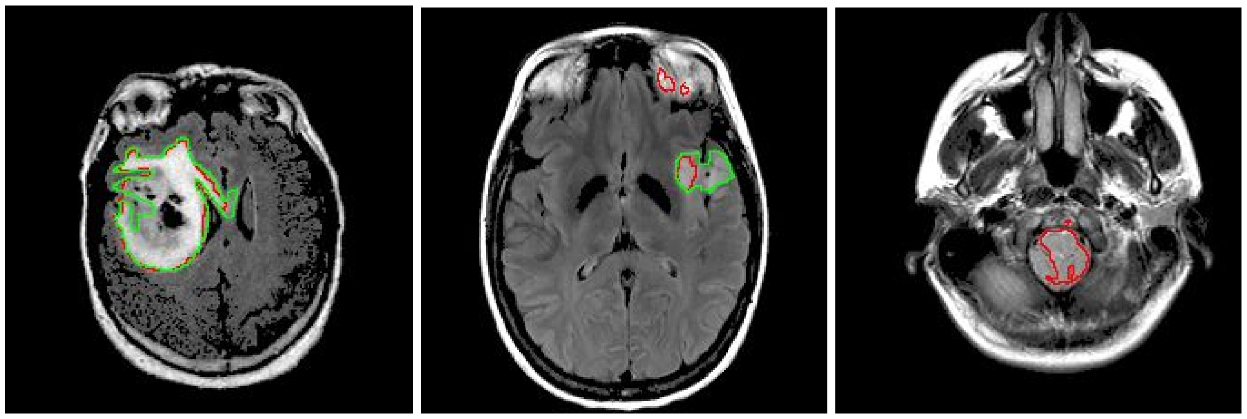

Figure 4: Examples of our baseline U-Net model output compared with ground truth. Green annotation represents the ground truth and red annotation shows what is output by our model after 50 epochs.

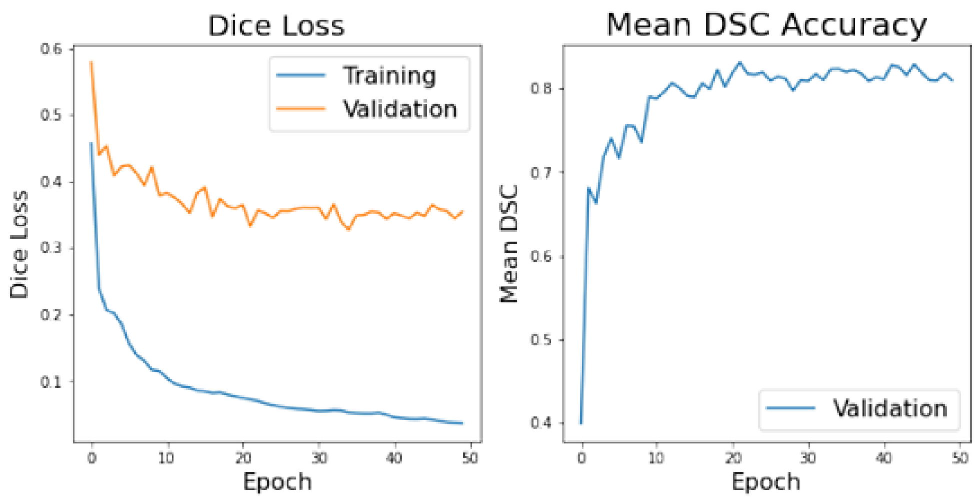

Figure 5: The left plot depicts the dice loss function as a function of epoch number for both the training (blue) and validation (orange) set. The right plot displays the average DSC accuracy as a function of epoch for the validation set.

Comparing our final results with the results obtained from the baseline U-Net structure alone, data augmentation seemed to improve our model's performance, yielding a best validation mean accuracy of 0.86 instead of 0.83 after 50 epochs. This time, a test set was also included to provide an unbiased evaluation of our model, but it was observed to have performed worse than both the training and validation sets, as shown in Figure 6. For this final model, we split our dataset into 80% training, 10% validation, and 10% testing, and used the parameters listed on the screen. Adam was selected as our optimizer, as it proved to be more accurate than stochastic gradient descent due to its adaptive training rate. While it was intended for our final model to include skull stripping, brightness adjustment, and elastic deformation, these methods were ultimately excluded from our final model as they appeared to worsen validation performance, yielding best validation accuracies within the 0.7-0.8 range. One potential explanation for this may be that all of the images in the dataset are relatively uniform in appearance; thus, introducing these different inputs train the model on features it will likely never see.

Again, the dice loss function decreased for both the training and validation datasets, with validation loss dropping from a value of 0.73 to 0.32, training loss dropping from 0.69 to 0.04, and testing loss dropping from 0.8 to 0.38 (Figure 7). The average dice coefficient accuracy, in contrast, increased and leveled off to about 0.94 for training, 0.84 for validation, and 0.78 for testing with the best validation mean accuracy of 0.8314 (Figure 7). The generalization gap observed for this final model with data augmentation implemented appears to have slightly decreased compared to the aforementioned baseline model.

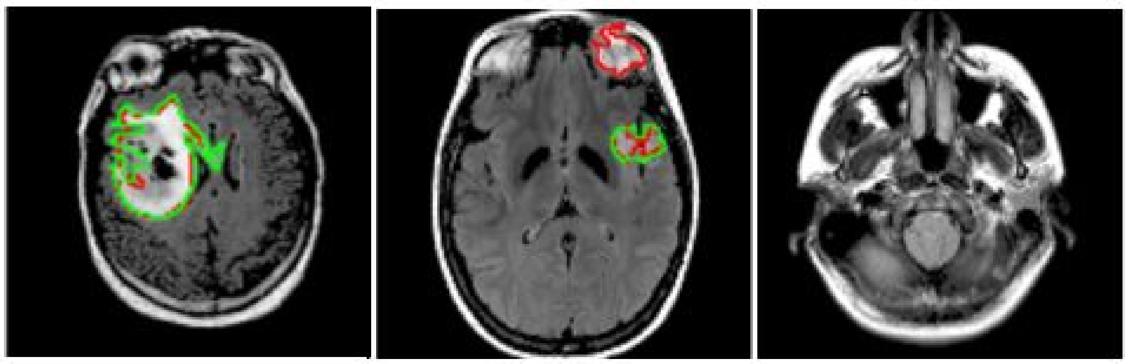

Figure 6: Examples of our model's output with data augmentation (scaling, rotation, horizontal flipping) compared with ground truth on the same MRI scans from Figure 3. Green annotation represents the ground truth and red annotation shows what is output by our model after 50 epochs.

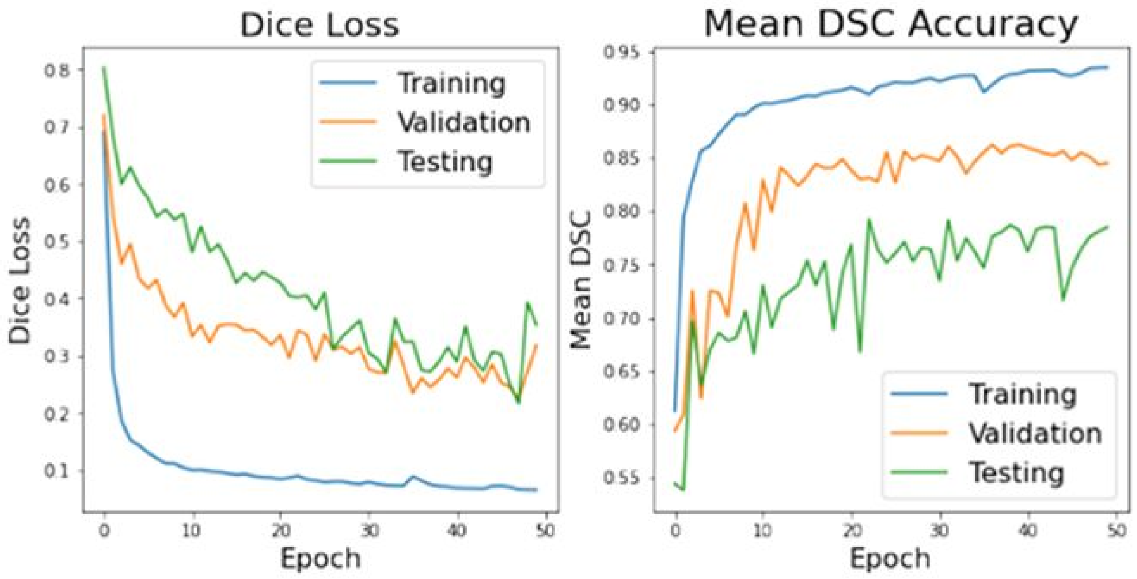

Figure 7: The left plot depicts the dice loss function as a function of epoch number for the training (blue), validation (orange), and testing (green) datasets. The right plot displays the average DSC accuracy as a function of epoch for the same three datasets.

FUTURE DIRECTION

We still believe that skull stripping would help the model's accuracy because we noticed that in some of the model's output, sections of the skull were marked as cancer. We believe that had we had the ground truth to train the skull stripping model on our dataset we would have had more success. However, it is time consuming to make such ground truths so alternatively morphologies can be used on the output of the skull stripping to fill in gaps where brain cancer was stripped out. This however, will not address that issue in the first image of figure 2 where a large section on the edge of the brain is missing. For cases like this, perhaps an algorithm can be used to predictively modify the output of the skull stripping model based on assumptions on what its general shape should look like.

REFERENCES

1. Rochester. (2020). Low Grade Glioma. Retrieved October 19, 2020, from https://www.urmc.rochester.edu/neurosurgery/services/brain-spinal-tumor/conditions/low-grade-glioma.aspx

2. Recht, L. (2019, July 25). Patient education: Low-grade glioma in adults (Beyond the Basics). Retrieved October 19, 2020, from https://www.uptodate.com/contents/low-grade-glioma-in-adults-beyond-the-basics

3. Niknejad, M., & Deng, F. (n.d.). Fluid attenuated inversion recovery: Radiology Reference Article. Retrieved October 19, 2020, from https://radiopaedia.org/articles/fluid-attenuated-inversion-recovery?lang=us

4. Buda, M. (2019, May 02). Brain MRI segmentation. Retrieved October 19, 2020, from https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation/metadata

5. Cirillo, Marco & Abramian, David & Eklund, Anders. (2020). What is the best data augmentation approach for brain tumor segmentation using 3D U-Net?.

6. Sankesara, H. (2019, January 23). U-Net. Retrieved November 08, 2020, from https://towardsdatascience.com/u-net-b229b32b4a71

7. Jordan, J. (2020, November 08). An overview of semantic image segmentation. Retrieved November 10, 2020, from https://www.jeremyjordan.me/semantic-segmentation/

8. Ronneberger, O., Fischer, P., &amp; Brox, T. (2017). U-Net: Convolutional Networks for Biomedical Image Segmentation. Informatik Aktuell Bildverarbeitung Für Die Medizin 2017, 3-3. doi:10.1007/978-3-662-54345-0_3

9. Brownlee, J. (2019, October 25). Difference Between a Batch and an Epoch in a Neural Network. Retrieved November 10, 2020, from https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

10. Brownlee, J. (2020, August 20). Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Retrieved December 10, 2020, from https://machinelearningmastery.com/adam-optimization-algorithm-for-dee

p-learning/

11. Lamba, H. (2019, February 17). Understanding Semantic Segmentation with UNET. Retrieved December 11, 2020, from https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

12. MaciejMazurowski. (n.d.). MaciejMazurowski/brain-segmentation. Retrieved December 11, 2020, from https://github.com/MaciejMazurowski/brain-segmentation/tree/master/skull-stripping