CS 4701 Project Proposal

Link to our Presentation:
https://drive.google.com/drive/folders/1UarPUIcbGaMEOPO0aKCLT5E-YJw9dI1G?usp=sharing

Title of Project:

Brain Tumor Recognition Software

Names/NetIDs:

Brian Arenas (ba328), Mahin Chowdhury (mac568), Abdullah Ridwan (aar233)

AI Keywords:

Convolution Neural Networks (CNN), DenseNet, Dropout Layers, Simulated Annealing, Overfitting, TensorFlow, Tensorboard, Fine Tuning Hyperparameters, Data Augmentation, Class Balance

Application setting:

To be used with brain imaging in medical settings to classify the different types of brain tumors. See description below.

Part 1: Project Description

The Big Picture

Cancer is amongst the leading causes of deaths worldwide, with brain cancer specifically compromising about 4.4% of those cancer deaths. In the United States, 15 to 20 people out of 100,000 are likely to have brain cancer [2], with each of them having a 36% 5-year survival rate [3]. Meanwhile, early diagnosis of brain cancer has continued to fall short. According to the World Health Organization (WHO), early detection is critical to improve patient's health and key to increasing their survival rate.

Hence, our goal is to use a convolutional neural network to classify brain tumors given MRI image data, using pytorch. The tumors that we will look at are the pituitary tumor, meningioma tumor, and glioma tumor.

What we planned to do

Our goal was and still is to create deep neural networks to classify brain images. Over the first months of the semester, we decided that using the DenseNet architecture (dense convolutional network) was the best course of action when it came to brain MRI classification.

We found that DenseNet was an ideal architecture because it improves upon convolutional networks that are very crowded and clustered. Other architectures like VGG and ResNet would yield lower accuracy and incorrect results for us due to their inefficient training, so we really needed a way to train our image data better (especially given that we could only go so far into this field without expert guidance and resources). This is where DenseNet came into the picture, as it makes training deep learning networks more efficient by shortening the connections between network layers. This was exactly what we needed to start yielding better numbers on our tests.

Something we spent some time on during the early semester was understanding if DenseNet would be specifically useful for brain MRI classification. We found that DenseNet was optimal because it closely connects network layers; each layer receives information from all the layers that come before it, each layer passes its own inputs to all the layers that come after it — which all means that the final layer to be used for our classification has influence from every single layer (in other words, much richer collection of brain image patterns), which is important for something as high stakes as brain analysis. This is what convinced us that DenseNet is the software that provides the best representation of images.

Another thing we planned to do was use transfer learning with our learning networks to save time and resources. With transfer learning, we would not have to train many ML models from scratch, since we could store information gained from images trained on earlier. In general, we saw examples over the course of the early semester that convinced us that transfer learning would be good for image categorization.

Key Aspect of AI: Convolutional Neural Networks

The main aspect of our AI program is using convolutional networks because they are the main way to categorize images. We use this to classify the different types of brain tumors -- pituitary, meningioma, and glioma — based on their images. We used densely connected convolutional neural networks (DenseNet) to achieve strong accuracy on our classifications with computer vision principles.

Part 2: Evaluation

Possible Roadblocks

When we were initially experimenting with the DenseNet, Shallow Convolution Network architecture and the Deep Neural network architecture, no matter what we tried, our accuracy would not go above 28% to 30%. Even doing trial and error for hyperparameters without doing any real finetuning, we were not improving. Furthermore, our naive understanding was that the more layers we add, the better our model would do. Hence, we moved from the DenseNet model, to a shallow architecture, to a deep architecture continuing 2, 3, 4, and 5 layers. However, as we added more layers, we saw our accuracy decrease! Hence, it was at this point that we had to assess our project and systematically determine what we can do to increase the accuracy score. In talking with each other and gaining more domain knowledge by looking at other papers that sought to classify brain tumor MRI images, there were 5 main roadblocks and questions, that we wanted to focus on, in our assessment of our project initially:

1. Do we even have enough data and is this a large issue as to why our model was not doing well?

2. Was there a data imbalance in terms of the classes with respect to the training, validation and testing dataset?

3. Are we using sub-optimal hyperparameters? While trial and error may be useful, it is also, in a sense, naive, because we don't know *where* to do trial and error from! Hence, the best idea would be to try to find the optimal hyperparameters for our model!

4. Maybe it's not a problem with the models at all but rather an issue with the preprocessing of our data. Specifically, is there a difference in using RGB images or Grayscale images?

5. Would the difference in the resolution of images impact our result?

Hence, for the next section, we will go into a deeper discussion on these questions, the tried solutions for these questions, and the answers we got form said solutions.

Thoughts on the Questions

First, let's look into the question of having enough data. As a rule of thumb, it seems that most deep learning models need a huge amount of data. In fact, the reason for subpar models in the machine learning industry seems to be the lack of clean, annotated and useful data. For example,

in my current research here at Cornell, we focus on medical text records and identifying patient information. However, creating a deep learning model to do this is hard because of a lack of access to already present data that has clearly annotated patient information within EHR records! In looking at our data, we only had 2870 brain tumor MRI images which we considered not to be substantial. Hence, we determined that we did *not* have enough data, and would have to do something to increase the amount of images we have, so the model can do better

Second, is a question of data balance. Specifically, note that there are 4 classes. The data can either be one of the three tumors or it can be an MRI image denoting no tumor. From learning more about deep learning from the brain MRI papers, previous classes, and other resources, we noted that the imbalance of classes within a dataset can be a huge problem. For example, say you have a model that should differentiate between dogs and cats, but you mainly have dog pictures, this model cannot be expected to do well on cat pictures! With this idea in mind, we performed some exploratory data analysis and noticed 2 main things. First, the relative class distribution amongst the training set and testing set was the *same*. This was good because that means that the training and testing set, proportionally, had the same number of classes with respect to the total amount of images. However, the real culprit here was something much more subtle - the class imbalance between the training set and *validation* set. Note, that initially during our preprocessing of the data, we would "stack" the data on top of each other. That means we add all non tumor pictures, then all pituitary tumor pictures and so on. Afterwards, when fitting and running the data on the model, we would denote a validation split of 0.2. However, in looking closer at the documentation for the function used, we noticed that the validation split takes the *bottom* 20% of the data! This is clearly an issue because that means that the validation data contains images from largely 1 class, which is a class that the training set will not see!

Third, is an obvious issue and was the first one that we thought of because it is something that we have run into in our previous classes. Not much needs to be said here other than it was something we needed to look into.

Fourth, was a question on the different color scales of the data and the reason why we were interested in this question even though it seems subtle is because of increased domain knowledge. Note, that while most CNN models would have RGB to be true - as in dog/cat classifiers and more, MRI images are essentially only black and white. Thus, the question arises if it is worth the increased computation time to look into an RGB image of a largely black and white image. Is there information to be gained? To determine this, we tried our different models on both grayscale and RGB images. A clearer description will follow in the next section.

Fifth and last was a question on the resolution. Does resolution even matter in this case? Note that the main reason we asked this question was because while we understand that increased resolution sizes provide more information, they're also much more computationally expensive

and time consuming. Pair that with the fact that google colab has a dynamic limit on the gpu accelerators, we had to get the most “bang for our buck” by determining an optimal resolution size.

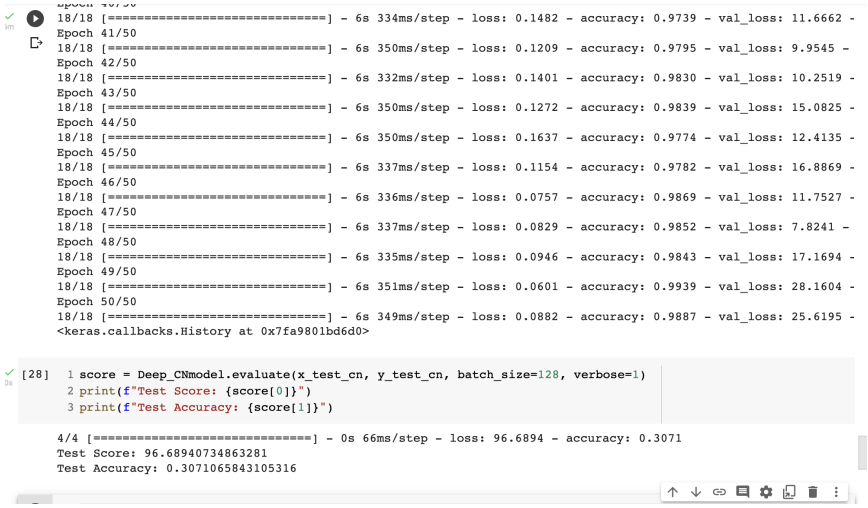
With a clearer description of the problems that we faced in assessing our project, let us now dive into the solutions that we tried.

Tried Solutions

To battle the issue of not having enough data, we looked into data augmentation. There are 2 routes that we took for this: using bing image search and using “pure” augmentation. For each one of our classes/different types of tumors, we queried for many images from Bing using a python package. The python package was able to get a substantial amount of images that we then appended/extended onto our training data. Furthermore, for data augmentation, we looked into “pure” data augmentation. The inspiration for this comes from our Computer Vision class in which we learned that we can increase the size of our data set by adding in zoomed in pictures, rotated images, horizontally and vertically flipped images, sheared images and more. Thankfully, keras has a module in which one can create such images called the ImageDataGenerator. Hence, we employed the ImagDataGenerator to create images with the aforementioned different transformations.

Now, as for the data imbalance, again, there was an imbalance between the training set and the validation set. Hence, to combat this we initially set shuffle to true when fitting the model onto the data. However, upon closer inspection, we realized that the shuffling of data happens *after* the data has been split which does not combat our issue. Hence, we utilized sklearn and both the shuffle and train_test_split function to manually shuffle our training set, and then split it into a training and validation set.

Third, for the hyperparameters, there were 2 routes that we took with advice from the TAs.. We utilized TensorBoard which is native to Tensorflow to allow us to visualize the change in accuracy and the validation accuracy per epoch. This also allowed us to see multiple other metrics and data about our model. In using tensorboard, we realized that while our accuracy increases, while the validation accuracy does not, pretty early on the epochs. This was an issue as if the validation accuracy does not increase but the training accuracy does, this is a clear indication of overfitting which will be detrimental to the model, and waste computation time for us. This became an issue as we can see in the image below where the model accuracy was incredibly high, yet the training accuracy was very low:



Hence, we utilized callbacks, *another* method naive to tensorflow. Specifically, we send the EarlyStopping callback which immediately stops and concludes model training when it is clear that the accuracy is going up but the validation accuracy stays the same — as a way to combat overfitting. Furthermore and more importantly, we used HPrams, another tool native to TensorFlow. HParams was made *for* fine tuning hyperparameters where the idea is that you provide the range of values that you want to test for the desired hyperparameters, and HParams goes through all possible combinations. Then, in a visualization similar to tensorboard, HParams can mention which combination of hyperparameters was the best.

Fourth in testing the color scales, we utilized a straightforward method of testing the different models on 10 epochs for both RGB and grayscale images,

Fifth, in addressing the question of the impact of resolution on our model's accuracy, we simply tried our the different resolutions at both the grayscale and RGB color scale.

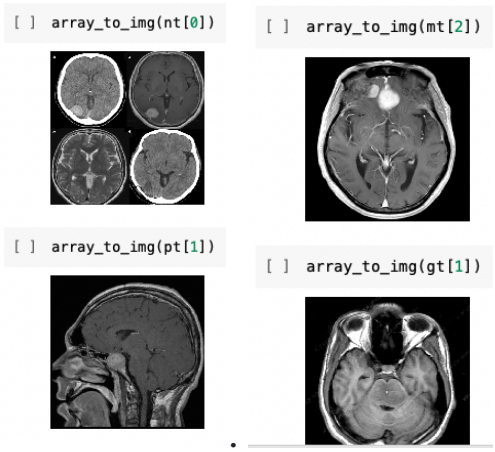
An honorable mention for something we were not able to do was use transfer learning. With brain imaging, we had the concern of initial and target layers not having similar enough patterns for the training to be considered relevant. Therefore, we did not utilize transfer learning in order to not encounter negative transfer.

In this next section we will provide what we learned from trying out these methods and the answers that these methods provided.

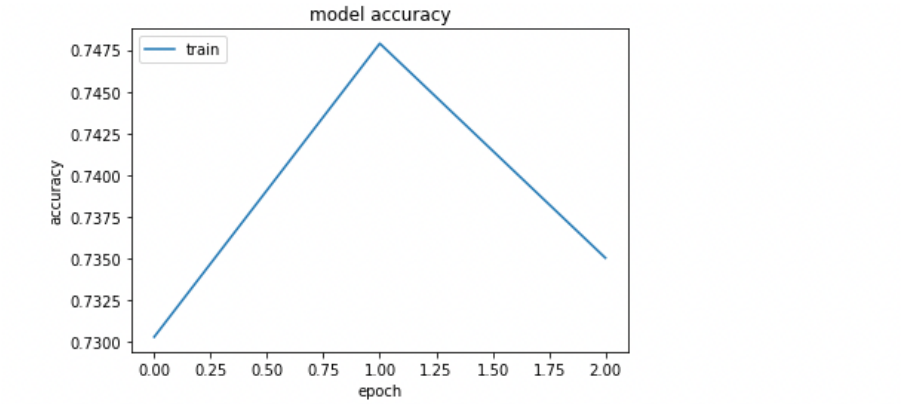
Answers (Kind of)

First, there were issues with the methods we used for data augmentation.

For the bing images, the model accuracy did not drastically. For example, we used the bing image searchers to get around 20 images of each type of tumor.
Via testing, the best query, we found, was something like “no tumor MRI image”. See below for an example of images that we acquired from Bing of “No Tumor”



While this increased the amount of data we had substantially, the issue was that the *quality* of these images were not up to par. In fact, some images had text, weird markings and colors, and overall the quality of the images would provide no help nor provide substantial information for the deep learning model. The accuracy of the DenseNet model increased from 30% to 35% in just 3 epochs as we can see below, yet we did not deem that substantial as the accuracy remained the same as the shallow and deep networks. Below is an image of the accuracy of just 3 epochs for DenseNet on the Bing Image augmented training data.



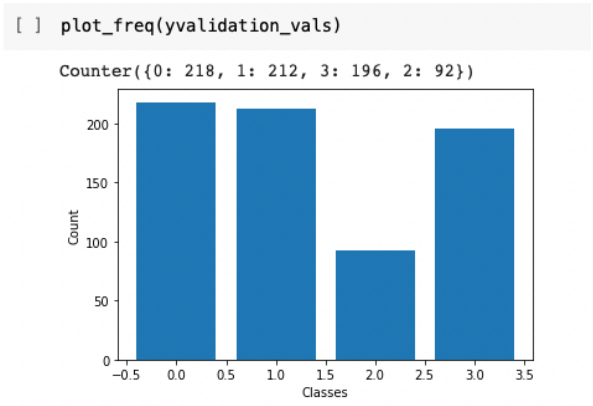
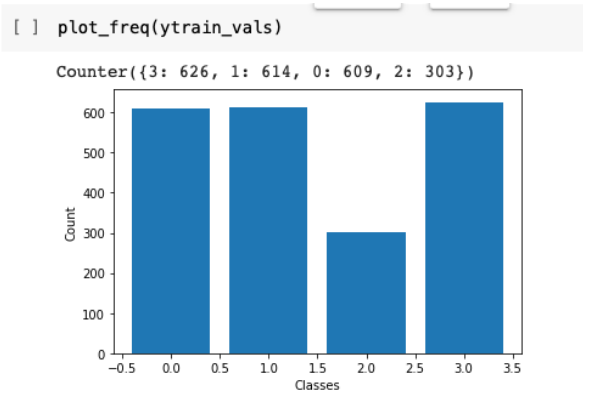
When trying the “pure” augmentation methods, our accuracy remained at 30%. While we initially performed all the different transformations, in gaining more domain knowledge, we realized we needed to be more thoughtful on *what* transformation we were doing. Note that this was the data augmentation parameters we were using initially:

Data Augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        )
datagen.fit(x_train_cn)
```

In the paper "Data Augmentation for Brain tumor Segmentation: A Review" the authors note that for transformations such as shearing, it is "rarely used to augment medical image data because we often want to preserve original shape characteristics". Furthermore, a transformation such as translation, which is one of the initial augmentations "may not be "useful" for all deep architectures—convolutional neural networks exploit convolutions and pooling operations, which are intrinsically spatially-invariant ". It seemed that the only ones that would provide some benefit was either horizontal flipping (not vertical) and slight zooming. The issue with vertical flipping was that the "up and down" parts of the image are not always interchangeable. Since some of the training images were either portrait or landscape in some instances, we decided to *only* use slight zooming in terms of data augmentation. After conducting this change in *how* we augment the data, we saw a substantial increase in the accuracy of our shallow CNN model!

For the data imbalance, we found that manually shuffling the data and then manually creating a training and validation split was substantial in increasing the model accuracy. Furthermore, with some quick data visualizations, we found that in doing such a random shuffle, we have a balanced number of classes in both the validation and training set. While the number of data points in the training set is obviously more than the count in the validation set, we see that the relative dispersion of the classes is the same in both sets:

Validation Set	Train Set
	

This is helpful and beneficial because then the validation set is similar to the training set, and the model can work better in improving the validation set after fitting on the training set. Note that shuffling the data allowed for *consistent* metric increases throughout all the models.

Putting it all together

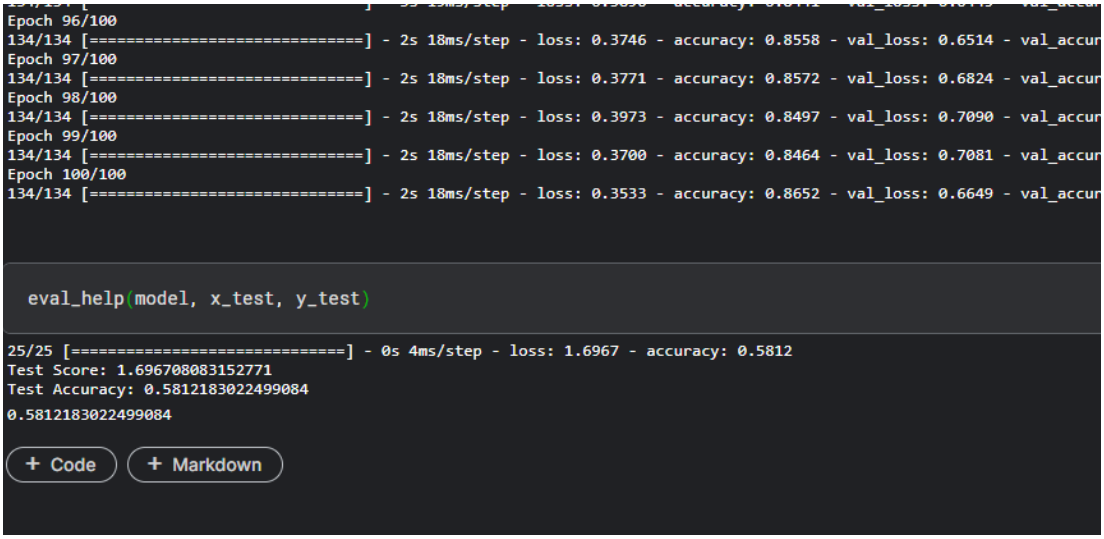
Thus, in answering these questions that arose from the assessment of our subpar models, we were able to learn much more about *what* we wanted the model to learn, and *how* it can best be learned. We ended up focusing mainly on the shallow neural network architecture because deeper neural network architectures were giving us lower test accuracy. We believe this may be because the dataset is simply too small even given the data augmentation. As noted above, we had multiple layered deep networks and ended up focusing only on the 3 layer network and comparing it to the "shallow" network.

Furthermore, we used "pure" data augmentation with minimal and specific transformations, shuffled the data to balance the training and validation set, and used 150x150 grayscale images. With regards to the hyper parameters and the failure of HParams: In getting guidance from the TA's and reading more about the different optimizers, we determined that the RMSProp optimizer was the best, and we played around with the different dropout rates via trial and error instead. Note that this trial-and-error method yielded the most significant response. We realized that the dropout rates were too high in this medical imaging domain. Hence, we decreased the dropout rates and saw dramatic increases, especially in the shallow Convolutional neural network Architecture. For example, changing the rates from 0.4, 0.5 to 0.2, 0.4 respectively, we found a solid 8% increase in test accuracy. The best increase, we saw, was with 0.2, 0.3 which yield an accuracy of 43% - 13% increase than before:

```
eval_help(model, x_test, y_test)
```

```
25/25 [==============================] - 0s 4ms/step - loss: 1.9895 - accuracy: 0.4315
Test Score: 1.9895168542861938
Test Accuracy: 0.4314720928668976
```

This was also when we started using the EarlyStopping Callback conservatively, because the idea is that EarlyStopping stops as soon as the validation accuracy doesn't change. However, mathematically, it may simply be that the model is "stuck" in a local minima. Hence, it needs time and epochs to be "unstuck". A solution for this problem of getting stuck in a local minima is simulated annealing and note that RMSProp implicitly performs simulated annealing. Hence, pairing that with higher epochs that *don't* have callbacks was expected to do well - and it did, especially for the shallow network!



How should we evaluate?

In determining how we should evaluate the success of our model, there is an important question we must first answer - is accuracy all that matters? While we initially believed this to be "yes", we quickly realized that this is not the case, especially as we increased our domain knowledge. While accuracy is important, there are other metrics that should be considered in the domain of brain tumor recognition such as precision and recall. Note that precision is the ratio of True Positives and the sum of True Positive and False negatives, and Recall is the True Positives divided by the sum of True Positive and False Negatives. These are important to consider because ideally we would not want to have a false negative - where the patient does have a tumor but the model says they do not. This may make physicians complacent under the guise that the model is accurate. Ideally we rather have false positives in the medical domain because in that

way, physicians would be prompted to do more rigorous tests to determine if the patient has a tumor. The goal is not to achieve substantial values to replace physician decision making, but instead aid decision making.

However, for our projects *specifically* this was hard to do as we do not have enough data points and as such our precision and recall scores were consistently low no matter the accuracy changes. Furthermore, we were not sure how to evaluate the scores as we believe an expert in the domain is needed or more domain knowledge is needed, to determine an appropriate range for precision and recall scores gives our dataset size. Hence, in evaluating our *specific* model, while we believe and concede that other metrics may be important in general, accuracy is the most important for our project given our dataset size.

Hence, as we increased our accuracy from around 30% to 58%-65% for epochs of 100 and up, we would say that our AI project was successful!

In general, we got extremely frustrated by the initial roadblocks and didn't realize how hard it would be to systematically go through all of the issues mentioned. We had to increase our domain knowledge by reading papers and more to even identify some roadblocks! Furthermore, we also didn't realize how difficult these computations are in general - we initially had *no* hardware accelerator, but then switched over to a TPU/GPU given the advice from a TA. Even then, we kept facing quota issues with google colab because their quotas are dynamic. As such, we had to bounce around different accounts and sometimes we would lose our work or crash notebooks because computations, usually in very deep networks, took up too much GPU ram and made us reach the weekly quota quicker. Lastly, the keras documentation and the TAs were very helpful in guiding us to relevant information that helped us gain more knowledge about building AI models and more!

References

[1] H. M. Bui, M. Lech, E. Cheng, K. Neville and I. S. Burnett, "Using grayscale images for object recognition with convolutional-recursive neural network," *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, 2016, pp. 321-325, doi: 10.1109/CCE.2016.7562656.

[2] H. Gao and J. Xinguo, "Progress on the diagnosis and evaluation of brain tumors." *Cancer imaging : the official publication of the International Cancer Imaging Society* vol. 13, no. 4, pp. 466-81, Dec. 2013. [Online serial]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3864167/. [Accessed Feb. 2, 2022]

[3] Cancer.Net, "Brain Tumor: Statistics" *American Society of Clinical Oncology*. [Online]. Available: https://www.cancer.net/cancer-types/brain-tumor/statistics. [Accessed: Feb. 2, 2022].

[4] J. Nalepa, M. Marcinkiewicz, andM. Kawulok, "Data Augmentation for Brain-Tumor Segmentation: A Review" *Frontiers in Computational Neuroscience:* vol. 13, 2019. [Online serial]. Available: https://www.frontiersin.org/article/10.3389/fncom.2019.00083. [Accessed March 15, 2022]

[5] Reveriano, Francisco. "The Advantages of DenseNet." *Medium*, A Journey Into Machine Learning, 24 Aug. 2019, https://medium.com/the-advantages-of-densenet/the-advantages-of-densenet-98de3019cdac.

[6] Tsang, Sik-Ho. "Review: DenseNet - Dense Convolutional Network (Image Classification)." *Medium*, Towards Data Science, 20 Mar. 2019, https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803.

[7] Gottapu, Ram Deepak, and Cihan H Dagli. "Densenet for Anatomical Brain Segmentation." *Procedia Computer Science*, vol. 140, 2018, pp. 179–185., https://doi.org/10.1016/j.procs.2018.10.327.