NeuroDet: Deep learning models for brain tumor classification

Hyeyeon Hwang
hhwang16

Alexandra Wong
amw11

Victor Youdom Kemmoe
vyoudomk

1 Introduction

We are interested in applying deep learning models to answer biological questions. Looking further into the types of biological questions we could focus on, we decided to focus on neuroscience and in particular, detecting types of brain tumors. According to the American Society of Clinical Oncology, this year, almost 25,000 American adults will have been diagnosed with primary cancerous tumors of the brain and spinal cord. As the tenth leading cause of death for adults, it's also estimated that 18,600 adults will have died from these tumors this year [1]. Obviously, identifying and diagnosing brain tumors is an incredibly important issue.

Typically, when an abnormality is seen on MRI or CT scan, the only way to confirm what type of tumor is present is to perform a brain biopsy, a very invasive diagnostic test that involves drilling through a patient's skull and snipping a tissue sample of the brain. This currently the only way to definitively determine which type of brain tumor is present. This is important to identify because different brain tumor subtypes are correlated with different levels of aggressiveness and treatment options. For example, a slow-growing tumor may not pose an immediate risk and could be easily removable by surgery. However, a more aggressive, fast-growing tumor may become quickly inoperable, forcing patients to try radiation and/or chemotherapy. If we can create a non-invasive method to diagnose patients via their imaging, we could save patients the pain and risk of complications that are associated with obtaining a brain biopsy.

To this end, our project is a classification task. We aim to classify brain MRI (Magnetic Resonance Imaging) images on whether there is a tumor present and additionally what type of tumor it is. In particular, we are interested in classifying the tumors into 4 classes (no tumor, glioma, meningioma, and pituitary tumor).

2 Data

For this project, we used a set of Brain MRI images sourced from Kaggle [2]. Tables 1 and 2 shows a decomposition of the dataset based on classes. Figure 1 shows a random sample of nine images taken from the training data.

Table 1: Decomposition of the dataset

Type	Glioma Tumor	Meningioma Tumor	Pituitary Tumor	No Tumor	Total
Training	826 ($\sim$ 28.78%)	822 ($\sim$ 28.64%)	827 ($\sim$ 28.81%)	395 ($\sim$ 13.76%)	2870
Testing	100 ($\sim$ 25.38%)	115 ($\sim$ 29.18%)	74 ($\sim$ 18.78%)	105 ($\sim$ 26.64%)	394

Preprint. Under review.

Table 2: Decomposition of the dataset for 2 classes

Type	Tumor	No Tumor	Total
Training	2475 ($\sim$ 86.24%)	395 ($\sim$ 13.76%)	2870
Testing	289 ($\sim$ 73.35%)	105 ($\sim$ 26.64%)	394

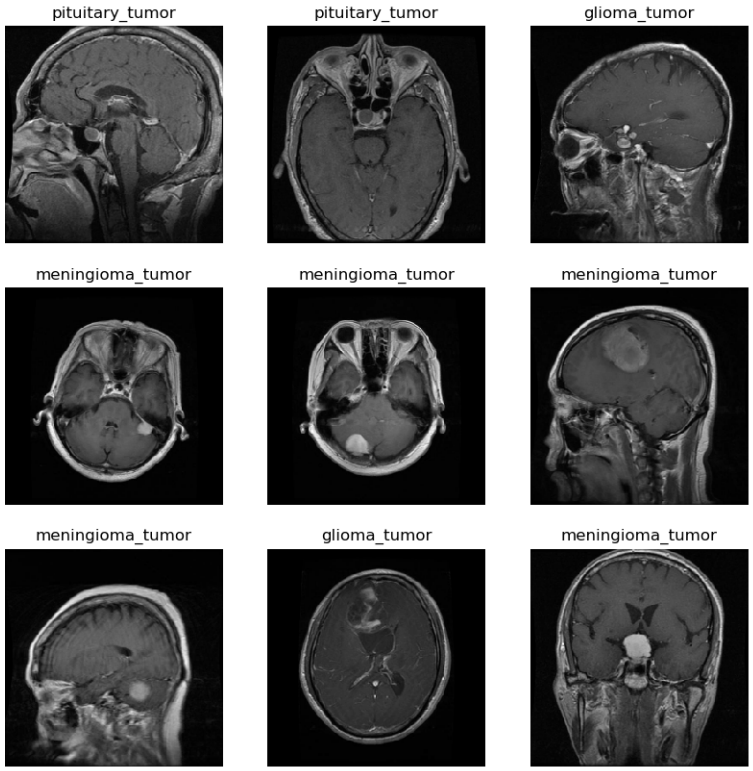


Figure 1: Random Sample of the Training Data

3 Methodology

3.1 Metrics

The main metric we used to assess the performance of our models is accuracy. Included in our results are figures showing the accuracy of the each type of model with different hyperparameters and ablation experiments, as well as the loss and training and testing accuracy curves across the number of epochs for the best performing models.

3.2 Loss function

We use Sparse Categorical Cross Entropy for all models [3]. It is a Tensorflow function that implements the Cross Entropy Loss function (see equation 1) without having to *one-hot* ecode labels.

$$\mathcal{L}_{\text{cr}} = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\Pr[f(x_i) = y_i]) \tag{1}$$

Where f is our model, y_i is the label, x_i is the input, and N is the size of the dataset.

3.3 CNN

The first architecture that we chose to experiment with is Convolutional Neural Network (CNN). With their *translation invariance* property (which is achieved by combining a convolution operation with a pooling operation), CNNs are generally good for image related tasks.

Our approach in designing a CNN for each of these two classification tasks (binary and quaternary classification) is to design a CNN architecture that will perform well on the quaternary classification and reuse that architecture to design a model that will perform well on the binary classification. This approach is based on the intuition that a model that is *good / average* on a task t should be able to produce good result on a task t' that is similar to t but has a smaller breadth.

CNN-4 Classes After a series of trial and error, we landed on the architecture depicted in our Figure 2b. The architecture is made of the following blocks:

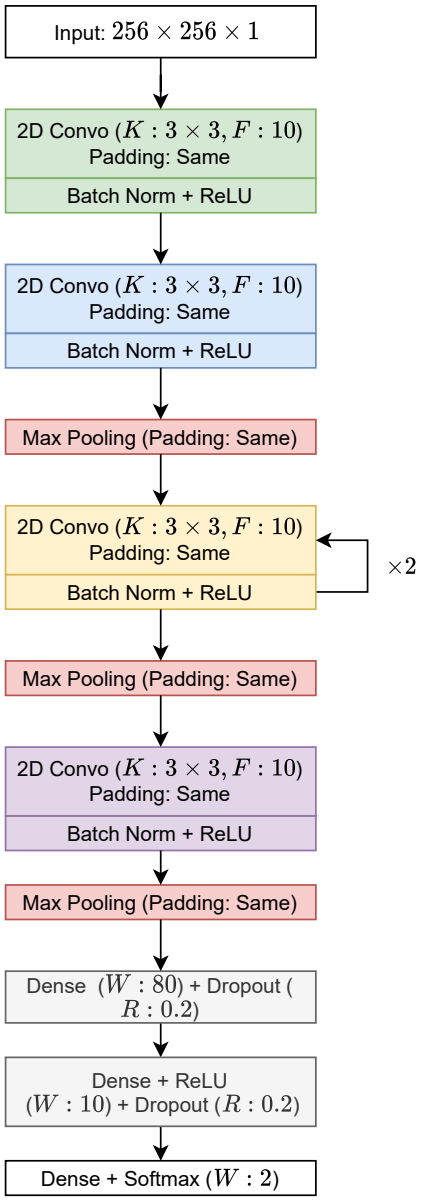
- four convolution layers that use a ten filters with a kernel size equal to three. After each convolution, we apply a Batch Normalization and a ReLU.
- three max pooling layers.
- two dense layers. We apply a ReLU on the output the first dense layer.

What is interesting with this CNN architecture is the use of a **Recurrent Connection** on a convolution layer (see yellow box on Figure 2b). Let c be a convolution operation and x be an input (in this case, an image). We define a recurrent connection over a convolution operation as follows:

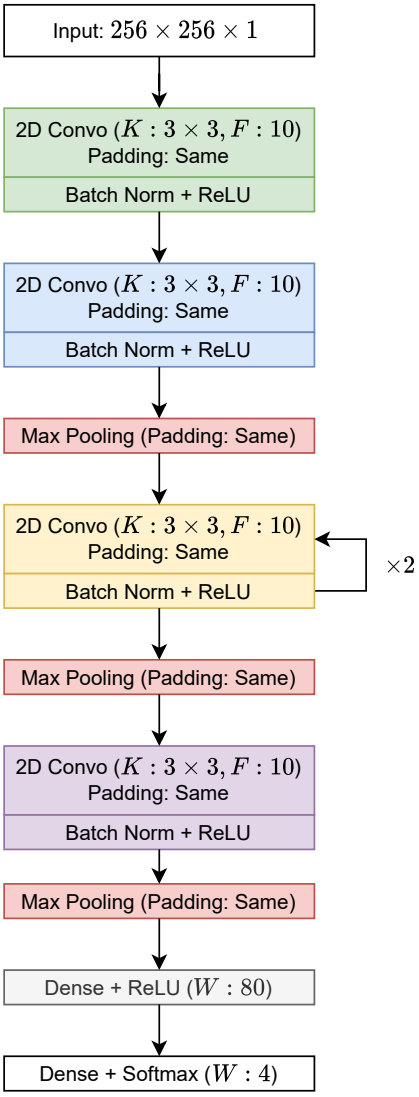
$$\mathsf{Output} = c(c(x))$$

In other words, we reapply the convolution c on its output. The best model that we obtained through our experimentation for this task uses that connection. The current hypothesis we have is that this type of connection produces a feature map that is more *clear* than the feature map produce by one pass, and therefore, help the model generalize better. However, due to time constraints, we were unable to dive deeper.

CNN-2 Classes a significant portion of the CNN architecture for the binary classification is based on the architecture of the quaternary classification, see Figure 2a. The main difference is in the set of layers that are appended after the last max-pooling layer. We used a set of three dense layers and the first two dense layers go through a dropout layer.

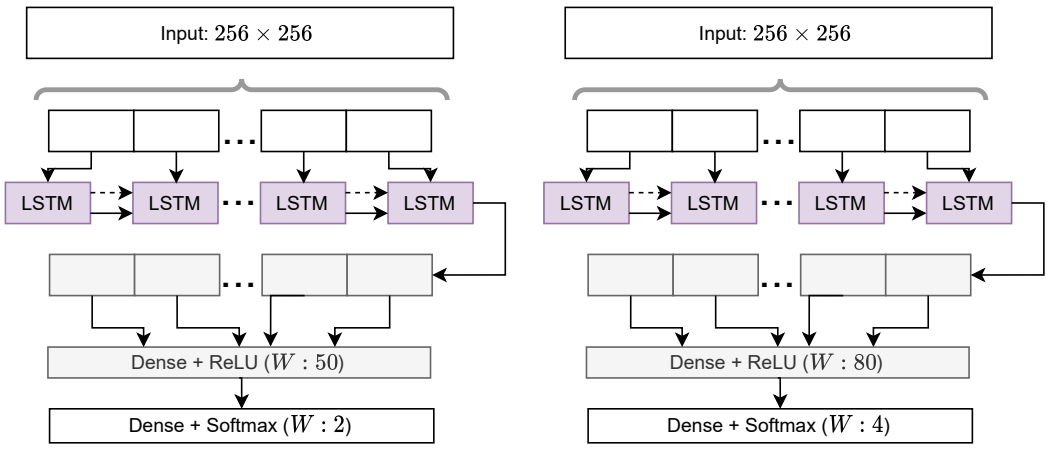


(a) Binary Classification



(b) Quaternary Classification

Figure 2: CNN Model Architecture



(a) Binary Classification

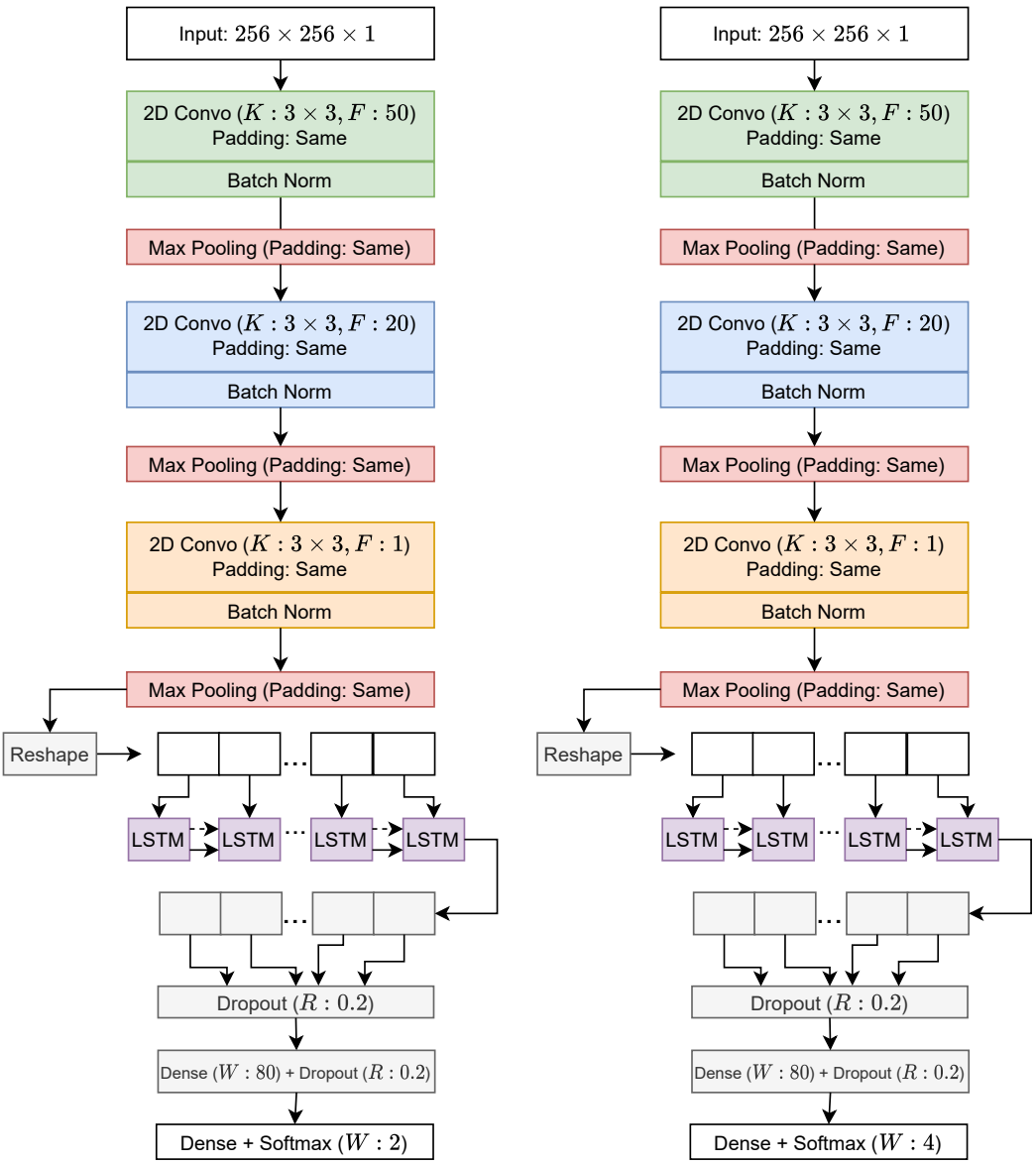
(b) Quaternary Classification

Figure 3: LSTM Model Architecture

3.4 LSTM

We read a study where researchers had reasonable success with the CNN-RNN architecture [4]. Since we decided to implement an LSTM instead of a vanilla RNN, we thought our analysis would not be complete without also comparing an LSTM by itself. Figure 3 shows the architecture of both the binary and quaternary classification models.

Our final LSTM models apply an LSTM layer, a Dense layer followed by a ReLU activation function, and finally a second Dense layer with a softmax function. The LSTM architecture for the binary and 4-class classification models are essentially the same. The difference between the 2- and 4-class model architecture is that the number of units for the Dense layers is 50 and 2 for the binary classification model and 80 and 4 for the quaternary classification model.



(a) Binary Classification

(b) Quaternary Classification

Figure 4: CNN-LSTM Model Architecture

3.5 CNN-LSTM

The alluded to in the previous section indicated that a combined architecture between a CNN and RNN could outperform other model architectures for multi-label image classification [4]. We decided that instead of only implementing an RNN, that instead we would try to implement a CNN-LSTM since the LSTM structure can help avoid some of the issues of vanilla RNNs. However, it should be noted that our architecture is different from the architecture presented in [4]. We first pass the input through a CNN. Then, we take the final output of the CNN and pass it through an LSTM (see figure 4). However, in [4], the authors use the CNN and the LSTM separately. The CNN is used to process the images and the RNN is used to process the labels of the images. After that, the output of the CNN and the RNN is combined through a Projection Layer.

3.6 Data Processing

The data processing operations that we applied on our dataset were straightforward:

1. **Rescale:** the first operation that we performed on our dataset was a re-scaling of images to the dimension 256×256. After exploring the dataset, we remarked that some images had a dimension of 1375×1446, whereas others had a dimension of 209×225. However, a significant portion of images had a dimension under 512×512. Hence, we opted to re-scale all of them to 256×256 since a larger dimension might have increased the amount of noise in small images and a smaller dimension might have reduce the clarity of large images.
2. **Shuffle:** we randomly shuffled both the training and test set.
3. **Normalize:** we normalize the pixels of each image by dividing their value by 255. The main purpose of this step was to reduce the variance between pixels by constraining their value in the threshold $[0, 1]$ and also improve the training process.

4 Results

Our code implementation is available at [5]. As you can see in the below Figure 5 comparing the model accuracy for both the two-label classification task as well as the 4-label classification task, the CNN had the highest testing accuracy in both cases. Additionally, the CNN-LSTM model was the second best in both classification tasks, followed by the LSTM in last place.

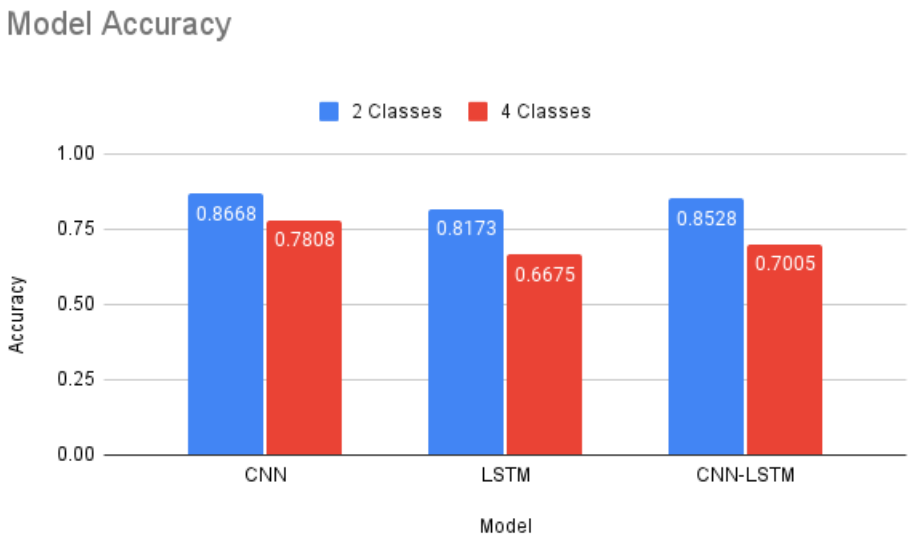


Figure 5: Model Accuracy Overview

In the following sections, we will explain how we found the best versions of each of these models and which hyperparameters accounted for the best increases in model accuracies.

4.1 CNN Configurations

In this subsection, we list the combination of hyperparameters and layers that we used to find the best CNN model for the binary and quaternary classification tasks.

Table 3: Constant hyperparameters and specifications for both 2-Class and 4-Class

Batch size	Optimizer	Num 2D Conv	Num Filter	Kernel Size	Padding
32	Adam	3	10	3×3	SAME

4.1.1 4-Class CNN

Table 4: Varying Hyperparameters for 4-Class CNN

	CNN-Plain (v1)	CNN (v2)	CNN (v3)	CNN (v4)
Num Epochs	30	30	40	40
Learning rate	0.005	0.001	0.001	0.001
Batch Norm After Convo	No	Yes	Yes	Yes
Num Convo layer	3	4	4	4
Recursion on 3^{rd} convo	No	Yes	Yes	Yes
Dense Layer	80 + Relu, 4	80 + Relu, 4	190, 80+Relu, 4	80 + Relu, 4
Dropout Layer	None	None	Yes (0.25) 1^{st} Dense	None
Augmentation	None	None	None	Yes (Horizontal, Vertical)

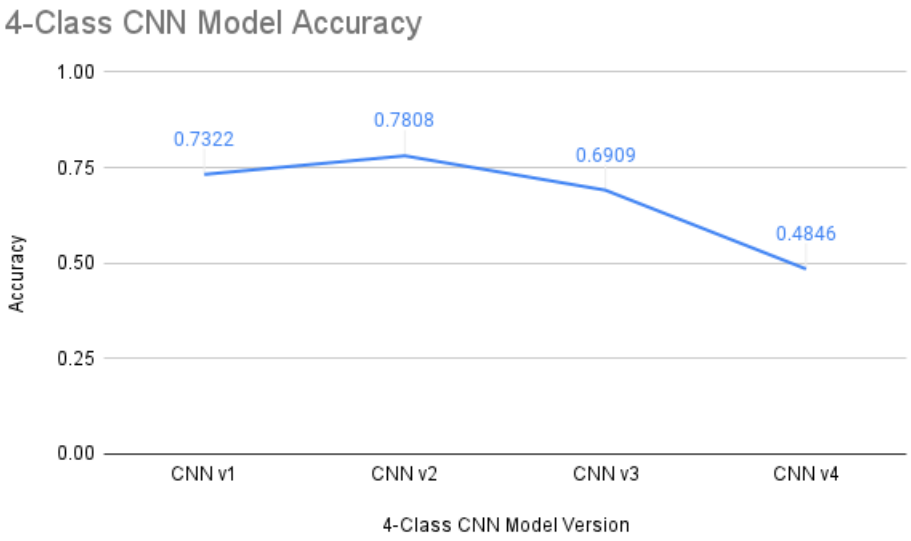


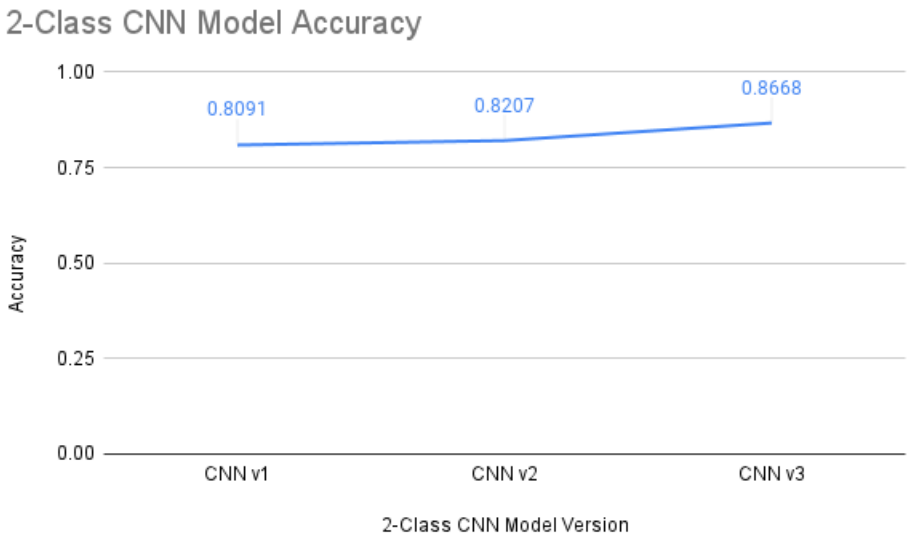
Figure 6

As can be seen in the above figure **??**, decreasing accuracy in the third and fourth versions of the CNN, adding a dropout layer and augmentation did not improve the model's accuracy. Version 2's hyperparameter combination was the best.

4.1.2 2-Class CNN

Table 5: Varying Hyperparameters for 2-Class CNN

	CNN-Plain (v1)	CNN (v2)	CNN (v3)
Num Epochs	40	50	40
Learning rate	0.001	0.001	0.001
Batch Norm After Convo	Yes	Yes	Yes
Num Convo layer	4	4	4
Recursion on 3^{rd} convo	Yes	Yes	Yes
Dense Layer	80, 4 + Relu, 2	80, 10 + Relu, 2	80, 10+Relu, 2
Dropout Layer	None	None	Yes (0.25) 1^{st} & 2^{nd} Dense
Augmentation	None	None	None



Here, in above plot, the best accuracy was in the third and final version of the CNN, which used a dropout layer.

4.2 LSTM

In this subsection, we list the combination of constant and varying hyperparameters and layers that we used tried for the LSTM model.

Table 6: Constant hyperparameters and specifications for 4-Class LSTM

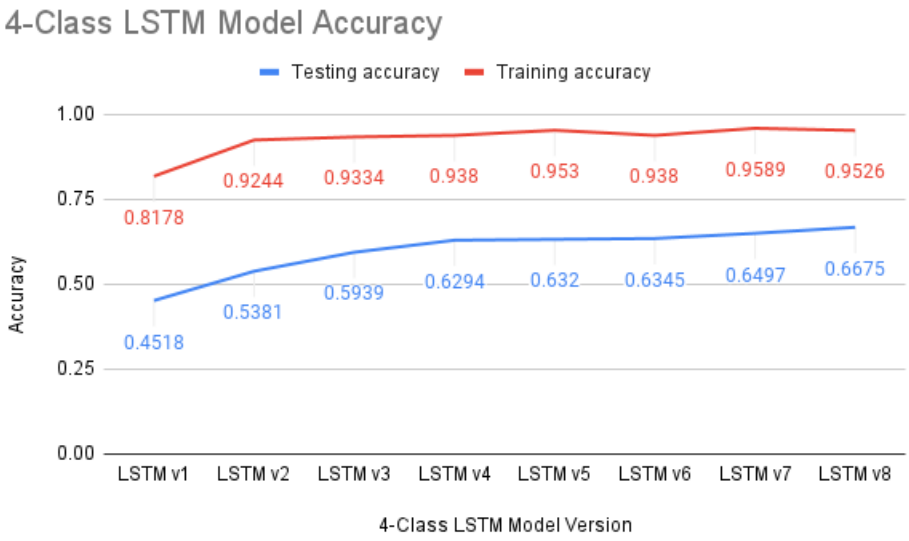
Batch size	Optimizer	LSTM units	Dense Layer 2 units
32	Adam	128	4

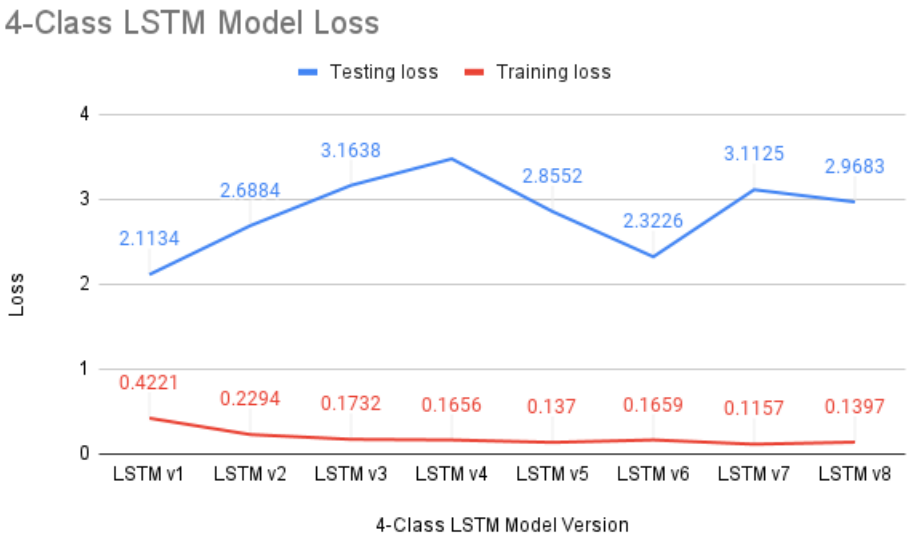
4.2.1 4-Class LSTM

Table 7: Varying Hyperparameters for 4-Class LSTM

	LSTM v1	LSTM v2	LSTM v3	LSTM v4	LSTM v5	LSTM v6	LSTM v7
Num Epochs	10	30	30	30	30	30	50
Num LSTM Layer	1	2	1	1	1	2	1
Learning rate	0.005	0.006	0.008	0.005	0.005	0.006	0.005
Dropout Rate (1^{st} Dense)	0.2	None	None	0.2	None	None	0.2
Dense Layer 1 units	80	128	80	80	80	80	80

	LSTM v8
Num Epochs	30
Num LSTM Layer	1
Learning rate	0.006
Dropout Rate (1^{st} Dense)	None
Dense Layer 1 units	80





The biggest difference in increasing the model accuracy across the versions tested in the above figures was increasing the epochs. In the comparison between testing and training loss and accuracy, we took care to make sure we did not sacrifice testing accuracy or loss for gains in training accuracy or loss, as we did not want to overfit our models.

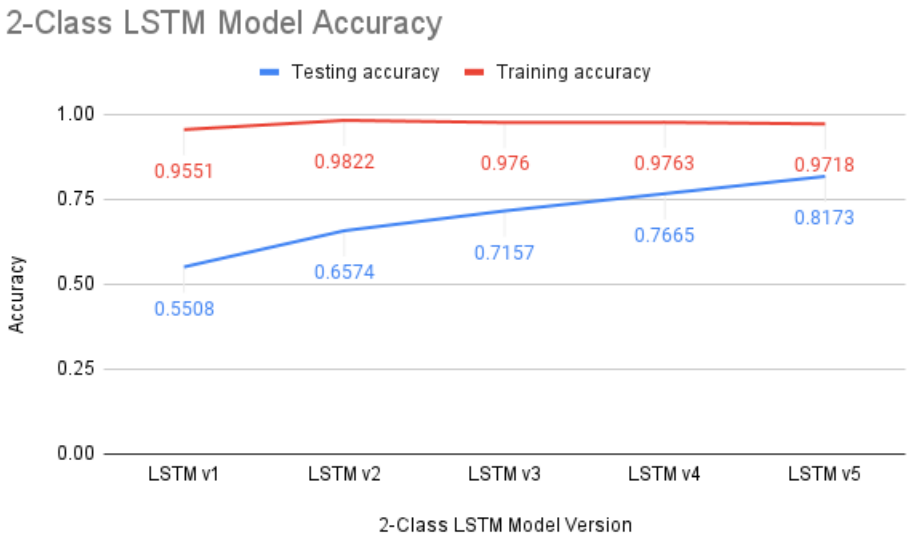
Table 8: Constant hyperparameters and specifications for 2-Class LSTM

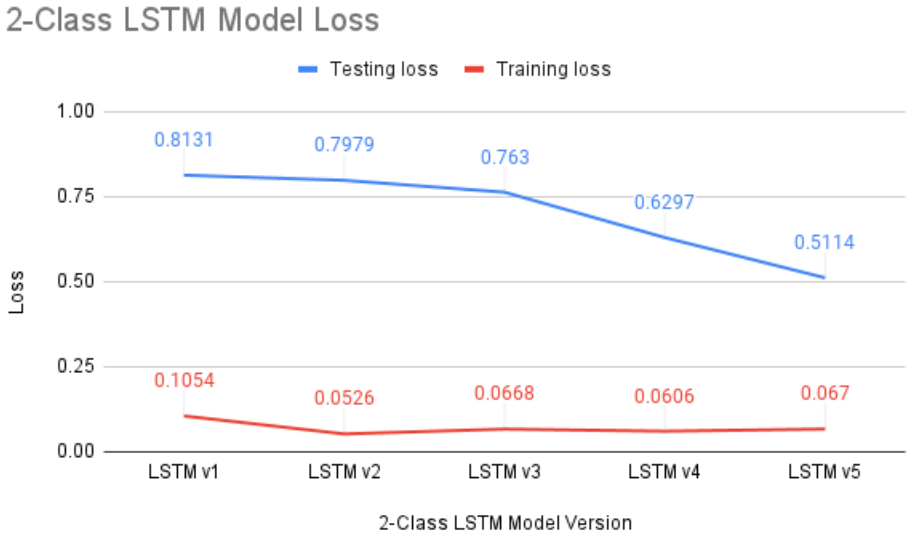
Batch size	Optimizer	LSTM units	Dense Layer 2 units		
32	Adam	3	10	3×3	SAME

4.2.2 2-Class LSTM

Table 9: Varying Hyperparameters for 2-Class LSTM

	LSTM v1	LSTM v2	LSTM v3	LSTM v4	LSTM v5
Num Epochs	10	30	30	20	50
Learning rate	0.005	0.006	0.005	0.004	0.005
Dropout Rate (1^{st} Dense)	None	None	None	None	None
Dense Layer 1 units	50	80	50	50	50





Similarly to the quaternary classification, the biggest difference in increasing the model accuracy across the versions tested in the above figures was increasing the epochs. As can also be seen in the comparison between testing and training loss and accuracy, we took care to make sure we did not sacrifice testing accuracy or loss for gains in training accuracy or loss, as we did not want to overfit our models.

4.3 CNN-LSTM

We selected five CNN-LSTM models that show the relationship between model performance and the three hyperparameters (number of epochs, learning rate, dropout rate) that influenced the performance the most. Generally, as the number of epochs increased, the model accuracy increased. Slightly increasing the learning rate and setting a dropout rate after the LSTM layer and first Dense layer also improved model performance.

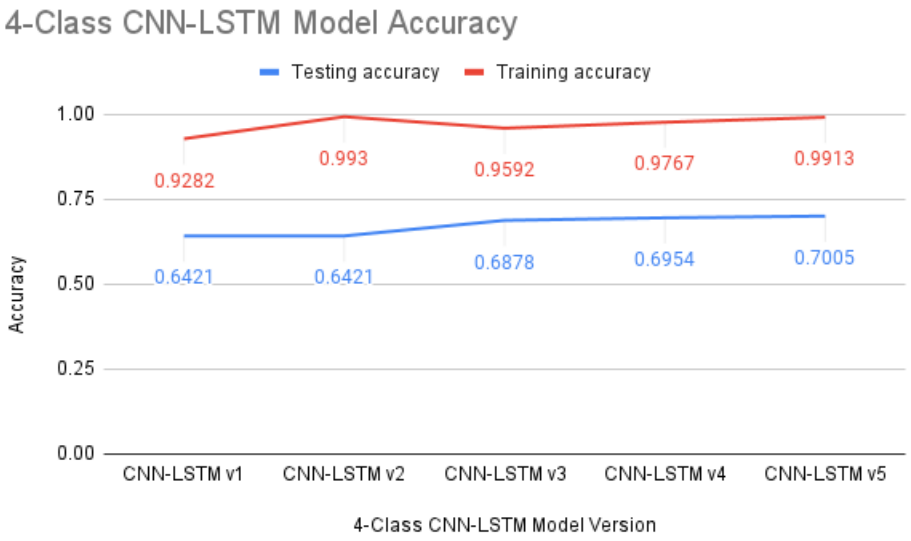
Table 10: Constant hyperparameters and specifications for CNN-LSTM 4-Class

Batch size	Optimizer	Num 2D Conv	Num Filter	Kernel Size	Padding
32	Adam	3	50, 20, 10	3×3	SAME

Num LSTM Layer	Num LSTM units	Dense Layer Units	Batch Norm	Max pool
1	128	80, 4	Yes	Yes

Table 11: Varying Hyperparameters for 4-Class CNN-LSTM (C-L)

	C-L v1	C-L v2	C-L v3	C-L v4	C-L v5
Num Epochs	10	20	15	30	50
Learning rate	0.002	0.002	0.005	0.005	0.005
Drop Rate(LSTM & 1^{st} Dense)	None	None	None	0.2	0.2



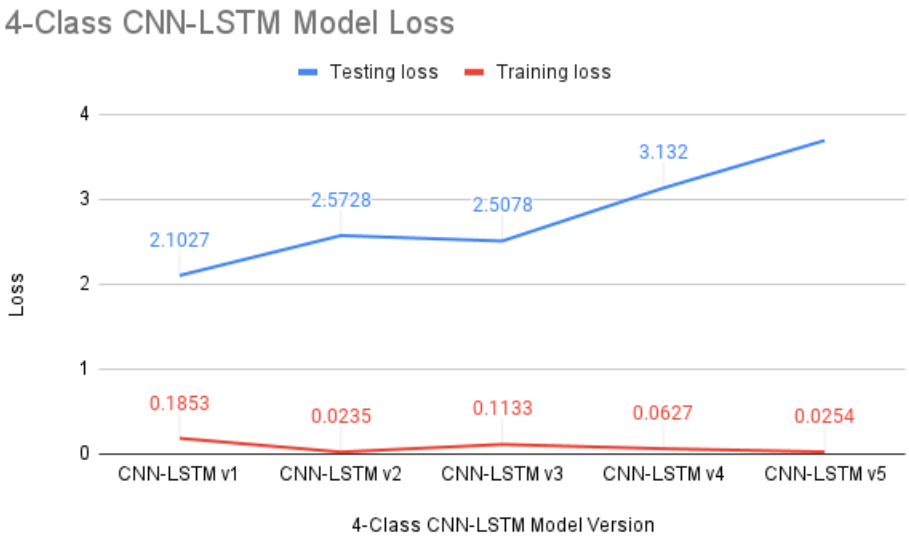
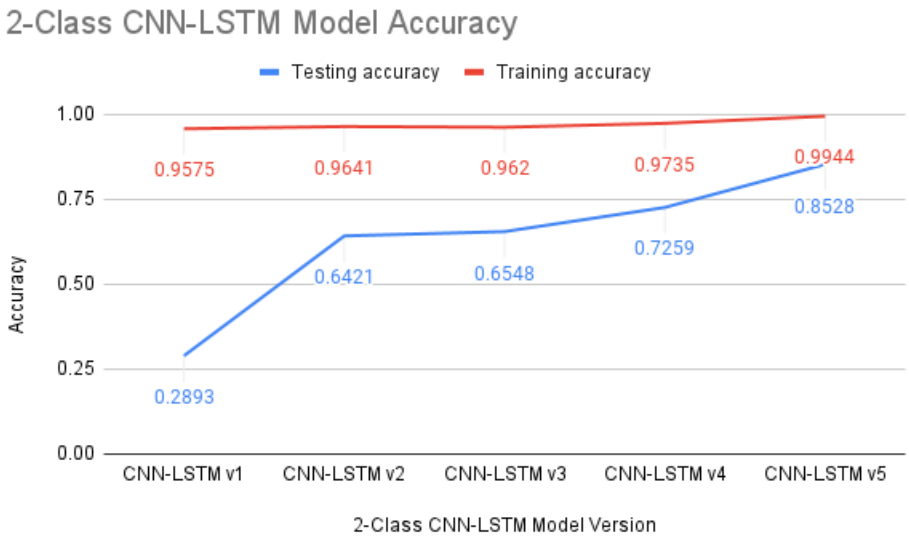


Table 12: Constant hyperparameters and specifications for CNN-LSTM 2-Class

Batch size	Optimizer	Num 2D Conv	Num Filter	Kernel Size	Padding
32	Adam	3	50, 20, 10	3×3	SAME

Num LSTM Layer	Num LSTM units	Dense Layer 2 Units	Drop Rate	Batch Norm	Max pool
1	128	2	0.2	Yes	Yes



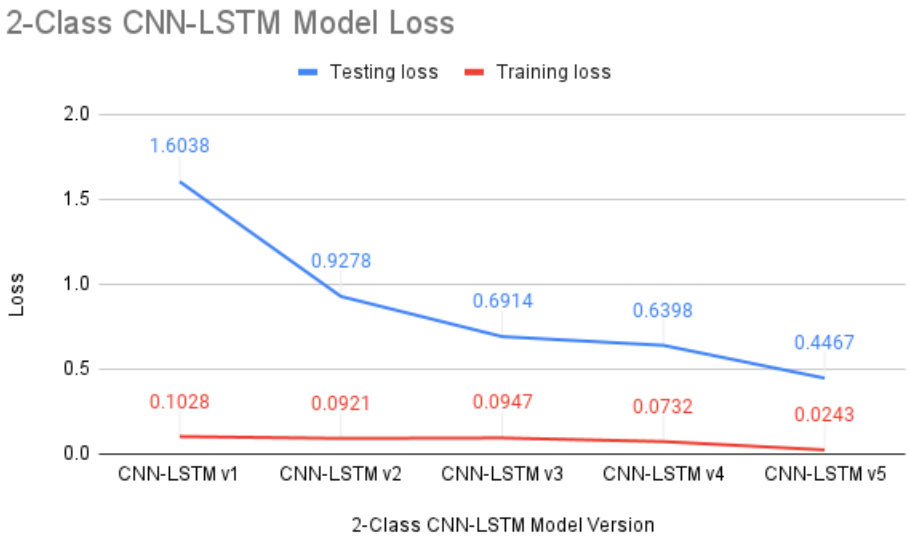


Table 13: Varying Hyperparameters for 2-Class CNN-LSTM (C-L)

	C-L v1	C-L v2	C-L v3	C-L v4	C-L v5
Num Epochs	20	20	20	20	50
Learning rate	0.01	0.005	0.005	0.006	0.005
Dense Layer 1 units	50	80	50	50	80

5 Limitations / Challenges

Implementing our models was the most challenging at first, but tuning them was another challenge. We first started with the 4-class classification models, and tuning the models to reach an accuracy of over 0.70 was extremely challenging.

We performed many hyperparameter tuning and ablation experiments but it was not exhaustive. Doing an exhaustive grid search of all hyperparameters for each model type may have yielded a better performing model, but due to limited time and computing resources we were not able to implement it.

As for limitations of the data, there was a disproportionate number of samples per class and the dataset was relatively small with only about 3300 images. (MNIST has 70,000.) Given the relatively small number of samples for an image classification task, our models performed decently well.

We also faced some health challenges for our team members. One of our team members was sick the entire last week leading up to the deadline, and another team member also developed some symptoms the last few days during this project.

6 Reflection

Our project was ultimately a success. Our initial base goal was to implement a CNN and RNN, target goal was to also implement a CNN-RNN, and stretch goal was to also implement a VAE. Although we have attempted it, we were not successful in implementing the VAE. With our CNN, LSTM, and CNN-LSTM implementations, we have met our target goal.

Our models more or less performed as we had expected them to. Based on our literature review, we expected either the CNN or CNN-LSTM model to perform the best, and as expected, the CNN model yielded the highest accuracies of 86% and 78%. We did not expect the CNN-LSTM to perform as poorly as it did on the 4-class classification task. For binary classification, however, the CNN-LSTM model was a close second with an accuracy of 85%. All models performed better

for binary classification, which we had anticipated since it is a simpler task to classify tumor and non-tumor images than to classify different types of tumors.

Our approach did not change over time, but we did leave out the VAE implementation (stretch goal), which we had attempted. If we could do our project over again, we would have more rigorous comparisons between the three models by using the same points of comparison, e.g. hyperparameter values. We are also interested in implementing a VAE and CNN-VAE instead of an LSTM and CNN-LSTM.

If we had more time, we believe we can further improve on all three of our models using an exhaustive grid search for hyperparameter tuning. Also, the number of epochs seemed to be one of the hyperparameters that influenced performance the most, but the maximum number of epochs we ran was 50 due to time and computing resource limitations. We would like to see if model performance improves with at least 100 epochs. We also want to try balancing the number of samples in each class as a preprocessing step and continue tuning our models. For our future directions, we are also interested in comparing our models with models presented in different literature using their benchmark datasets.

Our biggest takeaways from this project are that CNNs indeed perform well for image classification, CNN-LSTMs require an extensive amount of tuning to perform competitively with CNNs, and model tuning is a very resource intensive process.

In our course assignments, we used the MNIST and CIFAR datasets for image classification. With our final project, we were able to implement models on a different image dataset of MRI images. We also learned about the combined CNN-LSTM architecture, which actually did not perform better than the CNN.

References

[1] Brain tumor: Statistics. `https://www.cancer.net/cancer-types/brain-tumor/statistics`, 2021.

[2] Brain tumor classification (mri). `https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri`. Version 2.

[3] Tensorflow-API. `https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy`.

[4] Cnn-rnn: A unified framework for multi-label image classification. `https://arxiv.org/pdf/1604.04573.pdf`, 2016.

[5] Neurodet implementation. `https://github.com/amw14/NeuroDet`.

A Living Implementations

`https://github.com/aksh-ai/neuralBlack`
`https://github.com/Nupurgopali/Brain-tumor-classification-using-CNN`
`https://github.com/AryanFelix/Brain-Tumor-Classification`
`https://github.com/MohamedAliHabib/Brain-Tumor-Detection`