Comparison of Brain Tumor MRI Detection with Neural Networks vs. Traditional Machine Learning Models

Mohammad Ali Zahir ID: 40077619
Marwa Khalid ID: 40155098
Github Link: https://github.com/AliZ786/COMP-478

ABSTRACT

Magnetic resonance imaging (MRI) is a widely-used medical imaging technology in the healthcare industry that produces high-resolution images, which can be utilized for disease classification purposes. One limitation of MRI is its inability to differentiate between cancerous tissue and fluids due to the intricate nature of the human brain. As a result, the images obtained through the MRI need to be categorized as abnormal or normal by utilizing various image processing techniques. The purpose of this research study is to compare the effectiveness of three neural network models, namely ConvNet CNN, Multi-Layer Perceptron, ResNet50 and three traditional machine learning models – K-nearest neighbors, Support Vector Machines, and Random Forest in detecting brain tumors in MRI images.

I. INTRODUCTION

As technology has improved, humans have discovered more efficient ways to do tasks. These tasks include computer automation, video messaging, long-distance talking, etc. However, the most important would be the improvement of Artificial Intelligence, and more importantly Machine Learning. With these techniques, we have been able to analyze various trends and patterns in data, one of which is the analysis of cancer and tumor image datasets. Many of the databases of this nature contain black-and-white images with different segments being highlighted to show any anomalies that may be present. In this report, we wanted to compare which type of model would be best able to detect the

accuracy of brain tumor MRI (magnetic resonance imaging). We will compare three neural network models vs three traditional machine learning models. A neural network is a model in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain, also may be referred to as deep learning [1]. While a machine learning model is a model that gives computers the ability to learn without explicitly being programmed [2]. The three neural network models that we will use are the ConvNet CNN model, a Multi-Layer Perceptron model, and the ResNet50 model. While the three machine learning models that we will use are the K-nearest neighbors model, the Support Vector Machines model, and the Random Forest model.

II. RELATED WORK

There has been plenty of prior research done on the topic of determining the best model for medical image processing. In a research titled, Medical image analysis based on deep learning approach, we discovered that the CNN (convolutional neural network) is ideally suited for classification, segmentation, object detection, registration, and other tasks [3].

In another similar research, we could confirm the fact above even further as we found out that the CNN model archives a rate of 97.5% accuracy with low complexity compared with all other state-of-the-art methods [4]. As this is already a very high accuracy value, the goal of this report is not to achieve a higher value. Firstly, we will only be working on one single dataset, while the accuracy obtained was obtained on different and very large datasets. Secondly, similar to the first point, we will be using different models and try to compare the performance of each of them. The article does mention the accuracy of the CNN model, however, no accuracy for any of the other models was displayed, hence it would be difficult to classify arbitrary results as better. The goal of our report is to compare the results of neural networks vs. machine learning models for brain tumor MRI

detection. We will measure this by taking the accuracy value of all the models and observing which one gives a higher mean value at the end.

III. PROPOSED METHODOLOGY

A. Dataset

Our dataset contains a total of 3000 brain MRI images. This is then divided equally into two different folders, one for the images that contain a tumor, and another which does not contain the tumor. A randomly generated sample of ten each is shown in Figures 1 and 2 below.

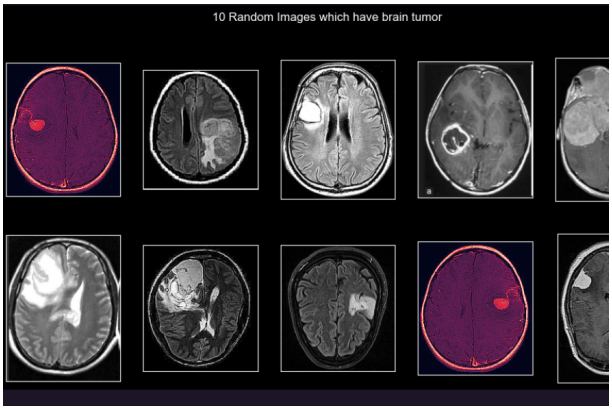

Figure 1: 10 random images that have a brain tumor

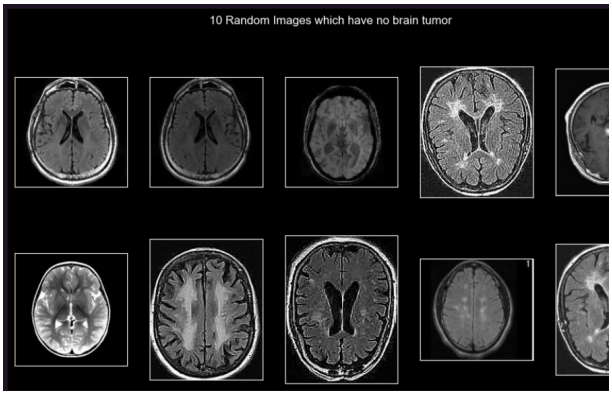

Figure 2: 10 random images that do not have a brain tumor

B. Preprocessing

As our data was not uniform, we had to apply some preprocessing steps to it.

1. Resizing the images

We wanted to make every image uniform, as it would then help us use them for the different models, which is why we decided to resize all the images in the dataset to 150 x 150 pixels.

2. Normalizing the features

We also wanted to normalize the features to be contained between the value of 0 and 1. To achieve this, we first loaded the resized images with our function called load_images. Once we did that, we divided each feature of all the images by 255, since these are all 8-bit images, and their pixel values lie between 0 to 255. Once that was finally done, we then returned the appended list.

3. Splitting the images into testing and a training set

We also split the dataset with the common 80/20 split using the scikit-learn train_test split method. From this, we saw that the number for both splits was fairly equal, which is ultimately what we wanted to achieve. Figure 3 displays the distribution of classes in both sets.

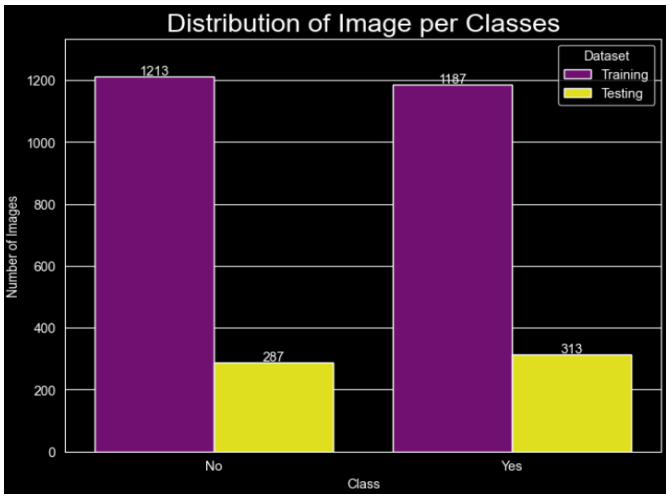

Figure 3: Distribution of Image per Class

C. Dependencies

Since the project is about comparing different neural networks and machine learning models, we had to import many Python data science libraries such as Scikit and Tensorflow. In addition to that, we also import libraries to display the data as plots such as seaborn and matplotlib, and the tabulate library to make the data cleaner for analysis.

D. Model Architecture

Since we have two different types of models, we will be explaining each model's architecture differently.

First, we will explain the neural network models.

1. The ConvNet CNN model

The CNN model connects multiple convolution layers together. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer [5].

In our specific case, the architecture of the CNN model is defined using the Sequential class from the Keras API. The model consists of several layers, starting with Conv2D layers with 32, 64, and 128 filters respectively, each with a kernel size of (3, 3) and ReLu activation function. These layers are followed by MaxPooling2D layers with a pool size of (2, 2) which helps reduce the spatial dimensions of the output feature maps. Finally, the flattened output is passed through two fully connected layers (Dense layers) with 128 units each, a ReLu activation function, and a final output layer with a single neuron and sigmoid activation function to produce a binary output.

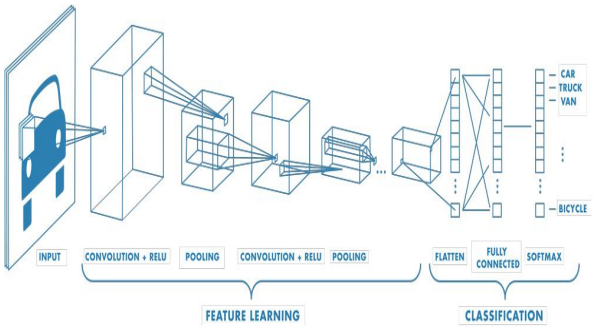

Figure 4: CNN Model Architecture [5]

2. Multilayer Perceptron

The Multilayer Perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs [6]. It learns hidden layer weights by adding bias and then using an activation function on that hidden layer.

Similar to CNN, the architecture of the MLP model is defined using the Sequential class from the Keras API. The model consists of a Flatten layer, which flattens the input images into a 1D array, followed by three fully connected layers (Dense layers) with 128, 64, and 1 units respectively, with ReLu activation function for the first two layers and a sigmoid activation function for the final output layer to produce a binary output.

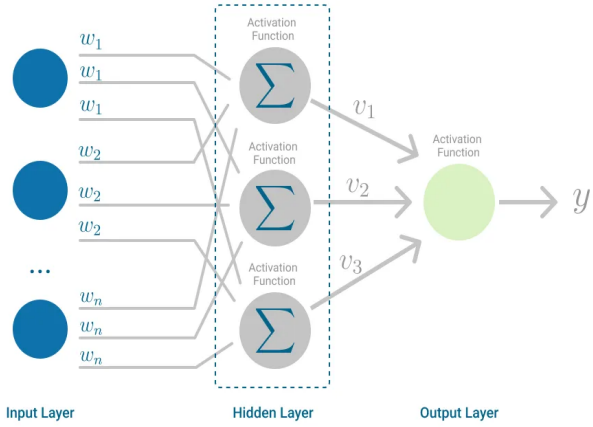

Figure 5: MLP Model Architecture [7]

3. The ResNet50 Model

This model is very similar to the CNN model explained earlier, however, it only goes a maximum of 50 layers deep. We load it with the ImageNet dataset and then pass our training data for it to compare the values generated by the ImageNet dataset with the dataset we provide [8].

In our specific case, it follows the same steps as the CNN, where we would first load the ResNet50 class from the Keras API. This is a pre-trained model that was trained on a large dataset of images (ImageNet). We would then have to remove the final connected layer which is the ImageNet database, and replace it with our database. To achieve this, we would set include_top to false.

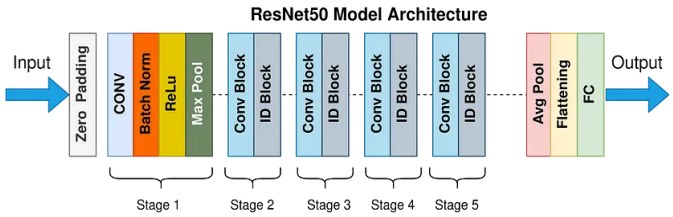

Figure 6: ResNet50 Model Architecture [9]

We now move on to our machine learning models.

1. k-Nearest Neighbors

The k-nearest neighbors (KNN) technique is a data classification approach that estimates the likelihood that a data point will belong to one of two groups based on which data points are closest to it [10].

This model falls into the supervised classification type, which means the labels are already known, as is the case with our dataset. The steps of this model are as follows:

1. **Load and normalize the dataset:** The first step in any machine learning algorithm is to load and prepare the dataset. In KNN, we also normalize the features to ensure that all variables have the same scale. We would then also split the features as well.
2. **Choose the number of neighbors (K):** The K in KNN refers to the number of nearest neighbors to consider when making a prediction. A small value of K can make the model more sensitive to noise, while a large value of K can cause the model to miss important patterns in the data.
3. **Fit the model:** The KNN model is fitted on the training data by storing the feature vectors and their corresponding labels.
4. **Predict the target variable:** Calculate the distance between the new data point and all the points in the training set, and select the K nearest neighbors. The predicted label is then the mode of the K nearest neighbors.
5. **Evaluate the model:** The final step is to evaluate the performance of the KNN model on the test set. This can be done using metrics such as accuracy, precision, recall, and F1-score. We can also use cross-validation to obtain a more reliable estimate of the model's performance.

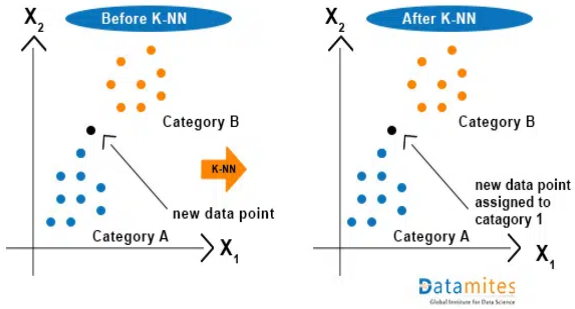

Figure 7: KNN example [11]

2. Support Vector Machines

SVM is a supervised machine learning model that uses classification techniques to solve two-group classification problems. They may categorize new text after feeding an SVM model a set of labeled training data for each category [12].

It creates a hyperlane, which is a linear line in the middle of the data, and then classifies values based on where they may pass depending on the line. Ideally, we want a 50/50 split. The steps for SVM are as follows:

1. **Load and normalize the images**: Ensures easier calculation of distance and makes everything uniform
2. **Flatten the images**: Because the SVM method requires input data in the form of 1D arrays, the 3D picture arrays are flattened.
3. **Create the SVM model**: The SVC function from the scikit-learn library is used to create the SVM model.

4. **Fit the model and predict the labels:** The SVM model is fitted on the training data using the fit method. Then, the predict method is used to predict the labels for the test set.
5. **Evaluate the model:** Compute the accuracy and other metrics from the model.

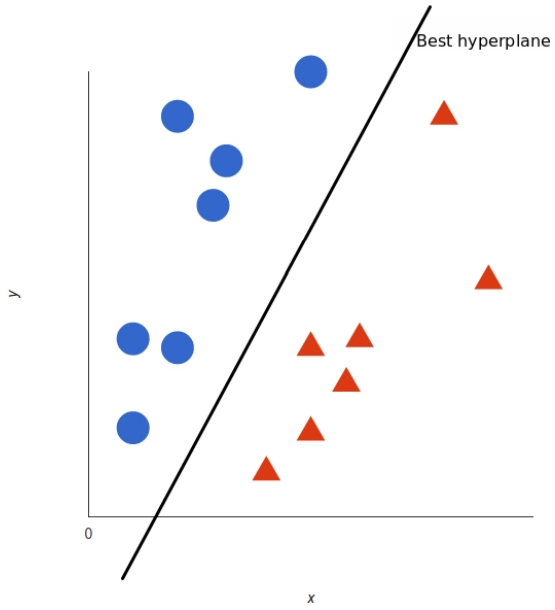

Figure 8: SVM Example [12]

3. Random Forest

Random forest is an algorithm that uses the results of several different decision trees and comes to a final polling decision on the class to put them in a class [13].

The trees are trained separately from each other, hence bias does not occur here and the result we get is then polled to give a final class. The steps for the random forest are as follows:

1. **Data Preparation:** This step involves loading the images, normalizing them and then ultimately flatting them for use in the Random Forest Model.
2. **Decision Tree Training**: Each decision tree is trained on a bootstrap sample of the data and a random subset of features, using a split criterion such as the Gini impurity or information gain.
3. **Combining Predictions**: Once all the decision trees have been trained, they are combined to make a final prediction. The mode of the class predictions of all the trees is taken as the final prediction.
4. **Model Evaluation**: The final step is to evaluate the performance of the random forest model using the test data. This involves computing metrics such as accuracy, precision, recall, and F1 score, and visualizing the results using techniques such as confusion matrices.

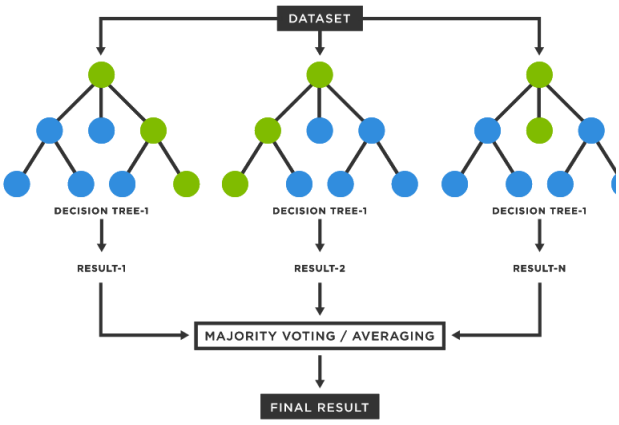

Figure 9: Random Forest Example [14]

IV. EVALUATION AND RESULTS

A. Hyperparameters

1. Neural Network Models

For training the chosen neural network models, we used two distinct architectures – Sequential model and ResNet50. The Sequential model utilizes the output of one layer as the input for the subsequent layer, making it ideal for models with a linear stack of layers like Multilayer Perceptron and Convolutional Neural Networks. In contrast, the ResNet50 architecture uses convolutional layers to extract image features, which are subsequently flattened and passed through another layer for classification. Listed below are the hyperparameters for the designated models.

1. ConvNet CNN Model and Multilayer Perceptron

I. *Number of Filters*: This numerical value signifies the quantity of characteristics that will be extracted for every layer. The ConvNet model assigns distinct values for each layer. Specifically, the first, second, and third layers have 32, 64, and 128 filters assigned to them, respectively.

II. *Kernel Size*: In a convolutional neural network, this value indicates the size of the window that moves across the input data to extract features. A 3 by 3 kernel size was assigned to each convolution layer.

III. *Activation Function*: This refers to the type of activation function applied to the output of a neuron. In the CNN model, the rectified linear unit (ReLU) activation function was applied to all convolutional layers which is defined as $f(x) = max(0, x)$. However, one of the Dense layers used the Sigmoid activation function which returns $\frac{1}{1+e^{-x}}$. On the other hand, in the Multilayer perceptron, the first 2 dense layers were applied the ReLU activation function, while the last layer was applied the sigmoid function.

IV. *Layer Size*: In Multilayer perceptron, this value signifies the size of the dense layers which are 128 and 64, respectively.

V. *Input Shape*: The input shape refers to the dimensions and number of channels. In both models, the input data is an image with a dimension of 150 by 150 pixels with 3 color channels.

2. ResNet50 Model

I. *Include Top*: This parameter decides whether to incorporate a fully connected layer at the end of the network or not. We set this parameter to false because we want to introduce our own customized layers.

II. *Weights*: This hyperparameter specifies whether to begin the weights with pre-existing values. We set it to None so that we can train the model from the start without any pre-existing weights.

III. *Input Shape*: In this particular model, we defined the input image size and the number of color channels as the input shape. Specifically, the input image is set to have a dimension of 150 by 150 pixels and 3 color channels.

IV. *Activation Function*: This pertains to the activation function utilized for the custom layers. One of the layers was applied the rectified linear unit (ReLU) activation function which returns *max(0, x)*. The other layer used the Sigmoid activation function which returns $\frac{1}{1+e^{-x}}$.

V. *Layer Size*: This hyperparameter signifies the size of the dense layers which are set to 128 and 1, respectively.

2. Machine Learning Models

We employed three different models for training our selected machine learning models, namely k-nearest neighbors, support vector machine, and random forest. K-nearest neighbors (KNN or k-NN) is a machine learning algorithm that predicts the class of a test data by analyzing the distances to its k-nearest neighbors. Support Vector Machine (SVM), on the other hand, is a supervised machine learning model that uses a hyperplane to categorize the data points into different classes. As defined by its name, random forest uses decision trees for improving classification and reducing overfitting. The hyperparameters for the specified models are outlined below.

1. K-nearest Neighbors

I. *Number of neighbors*: This parameter specifies how many neighbors should be considered when measuring similarity. In this case, we set it to 1, meaning only the closest neighbor is taken into account.

2. Support Vector Machine

I. *C*: This hyperparameter determines the balance between fitting the training data accurately and keeping the model simple. In this case, a value of 10 was chosen so that the model will prioritize fitting the training data accurately rather than generalizing to new data.

II. *Kernel*: This determines the type of kernel to be utilized. We selected the radial basis function as it's an effective choice for modeling complex nonlinear relationships.

III. *Degree*: This refers to the degree of complexity of the polynomial kernel function used. A value of 2 was set, indicating a second-degree polynomial.

3. Random Forest

I. *Estimators*: This hyperparameter refers to the number of decision trees, and in this case, we are utilizing a hundred decision trees.

II. *Max Depth*: Defined as the maximum number of levels each tree can have. We have chosen to set this value to 10.

III. *Random State:* refers to a seed value that guarantees that the outcomes of the model are repeatable. In our example, we chose to set the seed value to 42.

B. Evaluation Criteria

An evaluation metric is a quantitative measure that assesses the performance of a model. It helps to objectively compare the performance of different models. For our problem, we have chosen the specific evaluation metrics which are defined below.

1) *Accuracy:*

$$Accuracy = \frac{TP + TF}{TP + TF + FP + FN}$$

2) *Precision:*

$$Precision = \frac{TP}{TP + FP}$$

3) *Recall:*

$$Precision = \frac{TP}{TP + FN}$$

4) *F-scores:*

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

C. Confusion-Matrix

A confusion matrix is a table with two rows and two columns that are utilized to assess the performance of a machine learning classification model. The table's rows and columns depict the actual and predicted class instances. Each model has its own unique confusion matrix.

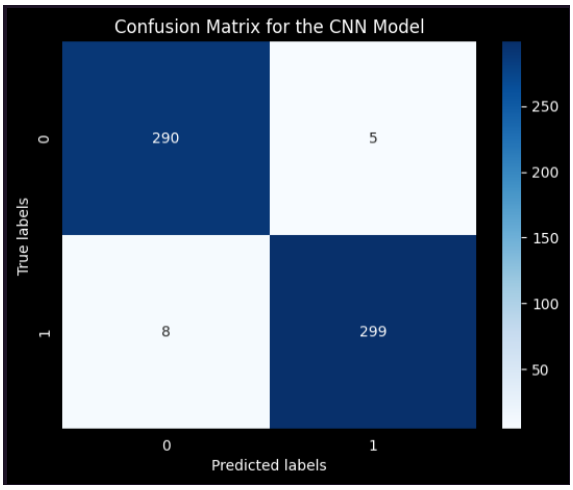

Figure 10: Confusion matrix for the CNN model

D. Results

After applying all the aforementioned techniques, we were able to achieve the following results.

Model	Accuracy	Precision	Recall	F1-Score
ConvNet CNN	97.84%	97.85%	97.84%	97.84%
MLP	97.01%	97.03%	97.01%	97.01%
ResNet50	77.24%	81.97%	77.24%	76.25%
kNN	97.18%	97.24%	97.18%	97.18%
SVM	97.84%	97.84%	97.84%	97.84%
Random Forest	93.18%	93.24%	93.19%	93.18%

Figure 11: Metrics for the tested models

Model	Accuracy	Precision	Recall	F1-Score
Neural Network	90.70%	92.28%	90.70%	90.37%
ML	96.07%	96.11%	96.07%	96.07%

Figure 12: Average for the tested models

For the values like precision, recall, and f1-score, we considered the weighted average in the results, although there was not a big difference between both values since the dataset was pretty evenly split.

For accuracy, we wanted to consider the validation set accuracy, as it would be the data that would be considered the test data in our case.

For Neural Network Models, we ran the models for 20 epochs. While the accuracy did not drastically change for MLP and CNN, it did drastically change for the ResNet model, as after 10 epochs the accuracy value was 49%, which at the end jumped to 77.24%. For the machine learning models, the results were initially in the 70% range, however, after some hyper-parameter tuning using GridSearch, we were able to achieve 90% plus values across the board for the machine learning models.

Comparing the values themselves, we can see that the machine-learning models fared better than the neural networks. This could be due to the fact that we found the best hyperparameters for the machine-learning models and not for the neural network models. If we just compared base to base, with regular default parameters, we would say that neural networks far and away had the better values.

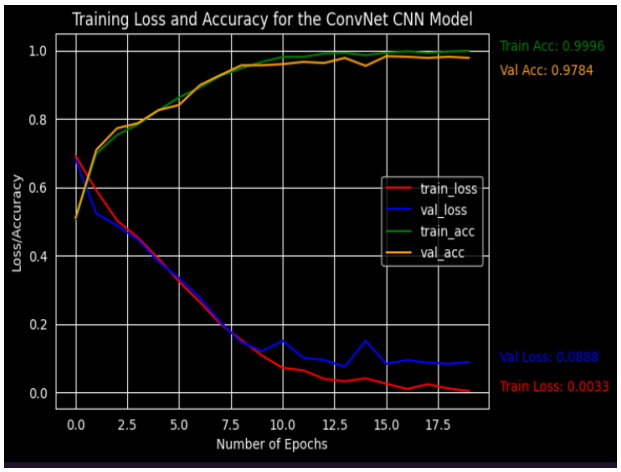

Figure 13: Loss and Accuracy of CNN for 20 Epochs

Figure 14: Loss and Accuracy of MLP for 20 Epochs

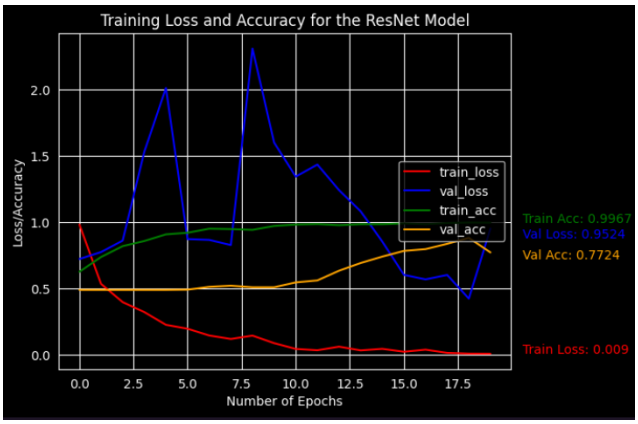

Figure 15: Loss and Accuracy of ResNet for 20 epochs

V. CONCLUSION AND FUTURE WORK

While we were able to say by our results that machine-learning models had better values than the neural network models, there were definitely many other factors involved. For example, if we ran the ResNet model for more than 20 epochs, we may have achieved higher values as the difference between 10 epochs and 20 epochs was a jump of around 30%. In the future, we should consider comparing all the default parameters first and then doing the hyper-parameter tuning to compare how much of a difference there is. We could also introduce new models, such as Generative adversarial networks (GANS) or even InceptionNet to see which model overall would be the best.

REFERENCES

[1] “What is ...,” Amazon, 1978. [Online]. Available: https://aws.amazon.com/what-is/neural-network/. [Accessed: 28-Apr-2023].

[2] S. Brown, “Machine Learning, explained,” MIT Sloan, 21-Apr-2021. [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. [Accessed: 28-Apr-2023].

[3] M. Puttagunta and S. Ravi, “Medical image analysis based on Deep Learning Approach,” *Multimedia tools and applications*, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8023554/. [Accessed: 28-Apr-2023].

[4] J. Seetha and S. S. Raja, “Brain tumor classification using Convolutional Neural Networks,” *Biomedical and Pharmacology Journal*, 21-Sep-2018. [Online]. Available: https://biomedpharmajournal.org/vol11no3/brain-tumor-classification-using-convolutional-neural-networks/. [Accessed: 28-Apr-2023].

[5] “What is a convolutional neural network?: 3 things you need to know,” *What Is a Convolutional Neural Network? | 3 things you need to know - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html. [Accessed: 28-Apr-2023].

[6] M. Rouse, “Multilayer Perceptron,” *Techopedia*, 30-Mar-1970. [Online]. Available: https://www.techopedia.com/definition/20879/multilayer-perceptron-mlp. [Accessed: 28-Apr-2023].

[7] C. Bento, “Multilayer Perceptron explained with a real-life example and python code: Sentiment Analysis,” *Medium*, 30-Sep-2021. [Online]. Available: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141. [Accessed: 28-Apr-2023].

[8] “Deep Network designer,” *ResNet-50 convolutional neural network - MATLAB*. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/resnet50.html. [Accessed: 28-Apr-2023].

[9] S. Mukherjee, “The annotated resnet-50,” *Medium*, 18-Aug-2022. [Online]. Available: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758. [Accessed: 28-Apr-2023].

[10] A. Joby, “What is K-nearest neighbor? an ML algorithm to classify data,” *Learn Hub*. [Online]. Available: https://learn.g2.com/k-nearest-neighbor. [Accessed: 28-Apr-2023].

[11] D. S. Team, “KNN algorithm in machine learning using Python,” *DataMites Offical Blog | Resources for Data Science*, 28-Sep-2022. [Online]. Available: https://datamites.com/blog/k-nearest-neighbor-knn-algorithm-in-machine-learning/. [Accessed: 28-Apr-2023].

[12] “Support Vector Machines (SVM) algorithm explained,” *MonkeyLearn Blog*, 22-Jun-2017. [Online]. Available: https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/. [Accessed: 28-Apr-2023].

[13] “What is Random Forest?,” *IBM*. [Online]. Available: https://www.ibm.com/topics/random-forest. [Accessed: 28-Apr-2023].

[14] “What is a random forest?,” *TIBCO Software*. [Online]. Available: https://www.tibco.com/reference-center/what-is-a-random-forest. [Accessed: 28-Apr-2023].