Brain Tumor Detection

1st Luís Silva
DETI
Universidade de Aveiro
Aveiro, Portugal
NMEC: 88888

2nd Pedro Amaral
DETI
Universidade de Aveiro
Aveiro, Portugal
NMEC: 93283

3rd Pedro Tavares
DETI
Universidade de Aveiro
Aveiro, Portugal
NMEC: 93103

***Abstract*—Brain tumor detection is a process performed every day by specialized doctors. Yet, it is also a classification problem whose objective is to analyze the presence or absence of tumors in the brain through various imaging techniques. This detection is crucial for the patient because if the tumor is detected in its early stages, the survival rate rises tremendously and with fewer complications throughout the treatment process. To develop our solution, a data set with 253 images from MRI scans of the human brain.**

***Index Terms*—brain, tumor, CNN, convolution, SVG, Transfer learning**

I. Introduction

Brain tumor detection is a useful process in the real world, mainly in edge cases where it is hard to determine if the tumor exists or not. This is true because, the earlier the tumor gets detected, the sooner it can be treated, leading to an easier and higher chance of survival for the patient.

This report contains 3 models which attempt to solve this problem using different machine learning approaches. In the next sections, we will first start by analyzing existing work regarding this theme Then the techniques associated with processing the data set will be explained. Next, the model's architecture will be detailed and the selection of its hyper-parameters will be described. Finally, a comparison will be made between the different models implemented.

This project was made in the context of the machine learning course at the University of Aveiro under the teacher Pétia Georgieva.

II. State of the Art [4] [5] [8]

When starting a project in machine learning it is always important to analyze previous work related to the theme of the data set. Most of the time, other people have already worked on similar data sets or even in the same data set. Given that our data set is from Kaggle there were a lot of projects that we could easily access about it.

Most of the previous work in this data set and theme used transfer learning which uses previously trained models to create another, using mostly VGG16 but also ResNet50. Besides transfer learning, convolutional neural networks are also very used in this theme since they are the most used for images in general. Outside of deep learning, Support Vector Machine is the most popular algorithm. Given this research, we decided that these would be the 3 models that would be implemented in this project.

Another factor apparent from our research is that most existing projects try to achieve better results using data augmentation to increase the size of the data set by artificially creating more examples.

III. Data Set Analysis

To train the models present in this project, a data set from Kaggle containing 253 images containing brain magnetic resonance images classified into 2 classes: positive and negative. In the images below we can see some examples from the used data set.

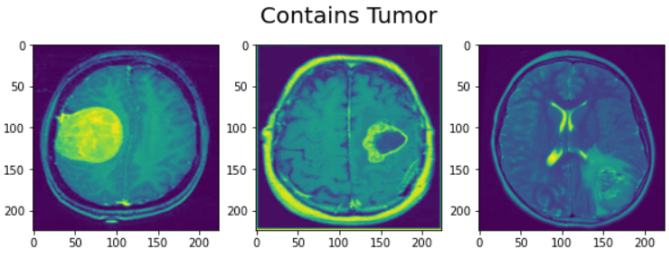

Figure 1. Examples of MRI with tumor

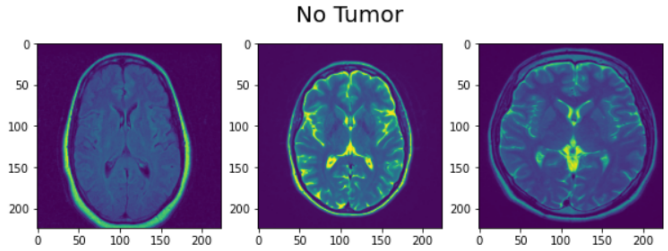

Figure 2. Examples of MRI without tumor

From these, we can observe that there are several differences between examples from the same class but there are certain details that are common that our algorithm should be able to distinguish. Our data set also presents a slight imbalance in the distribution of images between both classes.

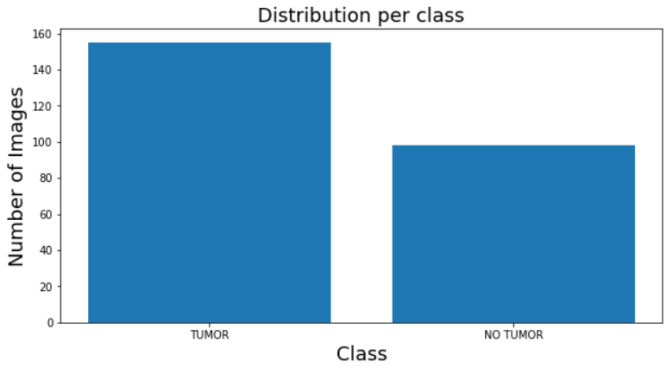

Figure 3. Distribution of images between the classes

Given this imbalance, data set balancing methods such as oversampling (repeat example from the smallest class) and undersampling (ignore examples from the largest class) may be beneficial. Besides the imbalance of classes, the data set used is small which may lead to some consistency problems. Since the amount of data is small, when compared to most data sets, the models will have fewer examples to train from. This problem could be solved through the use of data augmentation, which is a form of creating artificial data through the original data set and adding variety. This augmented data can be used as the training set used to train the model which should benefit the model by having more, and more varied data to pick from when training.

IV. Data Pre-processing

Before we use our data set in the models, its data must be prepared in a way that increases the performance of our algorithms. Given that we have different models, not every model used every single one of the implemented techniques using only the ones that made that model more effective.

A. Gray-scale Transformation

Our data set contains brain magnetic resonance images which are artificially given colors. This makes the color irrelevant for this project and therefore, When the images are read, they are always read in grayscale.

B. Cropping

Another technique that was used was cropping the image so that only the essential parts are left. This makes it so that when a future resizing happens less relevant information is lost. Despite this, the only model that seemed to benefit from cropping was the Convolutional Model giving the other 2 equal ao inferior performances when used.

C. Resizing

Resizing was another technique that was used in every model since, for the model to be trained, all the images given to him must have the same size.

D. Normalizing

Normalizing consists of converting every pixel to a value between 0 and 1 so that it becomes easier for the model to learn from the data. Despite this, it was not used in the Transfer Learning Model since it severely reduced its performance.

E. Shuffling

Finally, the data was also shuffled so that the model can learn more effectively from our data set. This is another mandatory step and therefore it happened in the preprocessing for every model.

V. Implemented Models

A. SVM Model

Support Vector Machine is a supervised learning model, it's goal is to separate the classes by defining a hyperplane through an N-dimensional space, with N being the number of features. Hyperplanes are essentially decision boundaries and while many could be drawn our goal is to choose the one that best fits our data while leaving room so that new, future data can be analyzed with confidence. This is achieved by choosing the one that maximizes the distance between two points of our classes.

Our implementation was based on the code presented in [10] and [11].

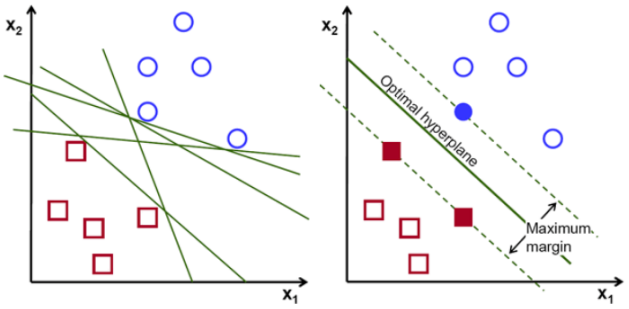

Figure 4. Maximization of margin when choosing a hyperplane

The support vectors that give name to the model are the points of each class that fall closer to the hyperplane, these influence the plane's position and orientation. Deleting or adding data points within the margin (before training) will result in a change to the resulting hyperplane.
The model can be tweaked by changing a few parameters

1) C: This parameter denotes the importance given to a misclassification of an example. The higher C is the higher the penalty. Increasing C will give more emphasis to fitting the curve to the training data which may lead to overfitting, decreasing C will put more emphasis on the generalization of the properties and thus can lead to a model that is better fitted to classify future examples.

2) Gamma: Determines how fast the similarity drops to 0 as elements get further apart, defines the influence of single training examples. Gamma is equal to

$$1/2 * \sigma^2, \sigma - variance$$

with and lower gamma means higher bias/variance and higher gamma means lower variance/variance. [9]

3) Kernels: Set of mathematical functions that map lower-dimensional data into higher dimensional-data.

B. CNN Model

A convolutional neural network consists of a series of layers. The input layer, hidden layers that perform convolutions which are essentially a mix of two functions to generate a third one and the output layer.

1) Architecture: In our CNN model, we applied zero padding and defined the activation function as leaky ReLU, a version of ReLU, and two convolutional layers to process the images. The first one applies 64 filters and the second one, 128, both with a kernel size of 4x4 and strides of 1x1, after both convolutional layers, a max-pooling layer was added with a kernel size of 4x4 and the same dimension for the corresponding strides. After the first max-pooling a dropout layer was added to ignore 30%of the data at random whose main objective is to prevent overfitting. After all that is done, the data is flattened and goes through two dense layers, the first being of the type ReLU and the second of type sigmoid. Finally, it is compiled using Adam's optimizer and the SparseCategoricalCrossEntropy. Having this in consideration, the data processing sequence is as follows:

Layer (type)	Output Shape	Param #
zero_padding2d (ZeroPadding2D)	(None, 228, 228, 3)	0
activation (Activation)	(None, 228, 228, 3)	0
conv2d (Conv2D)	(None, 225, 225, 64)	3136
max_pooling2d (MaxPooling2D)	(None, 56, 56, 64)	0
dropout (Dropout)	(None, 56, 56, 64)	0
conv2d_1 (Conv2D)	(None, 53, 53, 128)	131200
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 128)	0
flatten (Flatten)	(None, 21632)	0
dense (Dense)	(None, 512)	11076096
dense_1 (Dense)	(None, 2)	1026

Total params: 11,211,458
Trainable params: 11,211,458
Non-trainable params: 0

Figure 5. CNN Model Parameters

The first convolutional 2D layer has 36, the second has 64 and finally, the third one has 128 filters of size 4x4 with a horizontal and vertical stride of 1 pixel. Batch Normalization is made only once along with the results from the 36 filters applied in the first convolutional layer; Max Pooling of 4x4 pools, with horizontal and vertical strides of 4 pixels; Dropout Layer: ignores data and does this by setting to 0 20% of the values of the input at random;

Convolutional 2D layer with 64 filters of size 4x4 with a horizontal and vertical stride of 1 pixel. Max Pooling of 4x4 pools, with horizontal and vertical strides of 4 pixels; Dropout Layer: ignores data and does this by setting to 0 30% of the values of the input at random;

Convolutional 2D layer with 128 filters of size 4x4 with a horizontal and vertical stride of 1 pixel. Max Pooling of 4x4 pools, with horizontal and vertical strides of 4 pixels;

Flatten to turn all the data into a 1D data structure; Dense with sigmoid activation;

2) Activation Function: For the activation function, in the CNN, we used a ReLU, which stands for Rectified Linear Unit which takes the input and if it is positive, it keeps it, otherwise, it sets its value to zero.

3) Loss Function: To determine if the predicted values are close to the real ones, there is a need to represent that discrepancy. To represent that discrepancy we are using the binary cross-entropy method which is one of the most commonly used in machine learning.

4) Optimizer: In this model, we decided to use Adam as our optimizer given that, according to our research, it is the most used optimizer. It combines the advantages of two other optimizers (Adaptive Gradient Algorithm and Root Mean Square Propagation) and outperforms them, being more efficient and using relatively less memory. [6]

C. Transfer Learning Model

Recapitulating, from our research most of the existing work regarding this theme uses transfer learning which consists in using a model already trained with a different but related data set in our model, therefore, applying existing knowledge to a new problem. Among the existing trained models, most of the existing work used VGG16 so our Transfer Learning Model will also use it. Our implementation was based on a model from kaggle [3] although some changes were made.

1) Architecture: Building our model besides the VGG16, a global pooling layer to reduce the dimensionality of the VGG16's output, a dropout layer to ignore 20% of previous weights, and 3 dense layers were also added.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14714688
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 4096)	2101248
dense_1 (Dense)	(None, 4096)	16781312
dense_2 (Dense)	(None, 2)	8194

Total params: 33,605,442
Trainable params: 18,890,754
Non-trainable params: 14,714,688

Figure 6. Transfer Learning Model Parameters

2) Activation Function: In the dense layers added after VGG16, an activation function was needed. In the first 2, ReLU was used but in the final layer sigmoid was used which returns a value between 0 and 1 depending on the input.

3) Loss Function: In this model Sparse Categorical Cross-entropy was used which computes the cross-entropy loss between the real labels and predicted labels.

4) Optimizer: This model also uses the Adam optimizer like the previous model.

VI. Model Training

To be able to correctly obtain and evaluate a model, our data set needs to be split into sets with different goals. Given that we had a sizable amount of examples compared to the number of classes we decided to use the standard way of splitting our data into Training Set (60%), Validation Set (20%), and Test Set (20%). [7]

1) Training Set: This set has the biggest percentage of the original data set because this set will be used to train the model.

2) Validation Set: This set has the main function of testing our models across the iterations to validate our model's architecture and hyper-parameters during its development given that this data is not in the training set it is a good measure to see if our model is correctly generalizing and is not getting over-fitted. When the architecture and the hyper-parameters are decided they will be used to train the final model alongside the training set.

3) Test Set: This set contains data that the model has never seen in the development phase and neither in the final training and it is used to make a final evaluation of the implemented model.

A. Data Augmentation

Data augmentation is a method that allows generating a bigger amount of data from the original data set. This generation can be performed through a variety of transformations, in particular, random rotations, symmetry along horizontal and vertical axes, scaling, and brightness, among others. This method can be used for model training since it expands and increases the variety of the training set which, in theory, should make the model more flexible to slight changes to the real images. In our solutions, we used data augmentation in the SVM model. Data augmentation was tested in the CNN and Transfer Learning models as well, but the results were not as satisfactory as the model without data augmentation.

B. Oversampling and Undersampling

As previously referred to, our data set is imbalanced, and therefore an undersampling and an oversampling approach were used but the only positive result was applying oversampling in the SVM Model. The performance of the other 2 models was unaffected or decreased by these methods.

C. Results

Given the models and techniques described above, in this section, the results of the implemented algorithms will be detailed.

1) SVM Model Validation: Data augmentation is an interesting approach because our data set is small and because there is a noticeable discrepancy between the number of examples of class yes and class no as seen in fig. 7.

Figure 7. Class distribution in data sets without oversampling

Applying data augmentation and producing examples of class no and class yes at a rate of 2 to 1, not only provides more examples to train and validate our model but also balances out the classes' frequency, this is called oversampling. In fig.8 we can see the effect that this process had on our class distribution.

Figure 8. Class distribution in data sets with oversampling

Analyzing the confusion matrixes for the model, displayed in fig.9 and fig.10, we notice a drop in performance from classifying examples from the testing dataset to classifying examples from the validation dataset which is, although not desirable, expected as it is now dealing with examples it has never seen before.

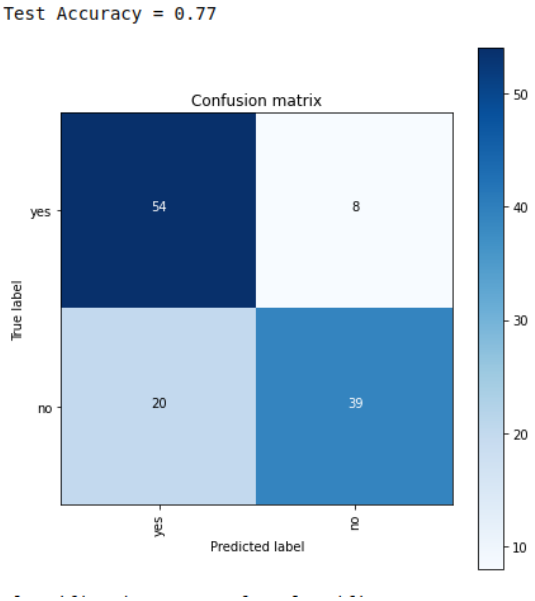

Figure 9. Accuracy in validation data set with oversampling

Table I refers to the models evaluation.

Table I
TRANSFER LEARNING MODEL STATISTICS

Train Accuracy	0.77
Test Accuracy	0.77
Precision	0.78
Recall	0.77
F1 Score	0.77

2) CNN Model Validation: In the following images, we can see the evolution of the model training throughout all its epochs. And as we can see, the divergence between the training and validation set, both in accuracy and loss values are relatively similar, meaning that the model is capable of generalizing and is not overfitting.

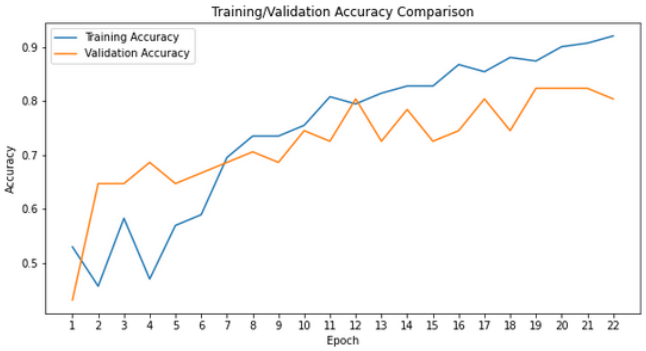

Figure 10. CNN Model Accuracy Comparison

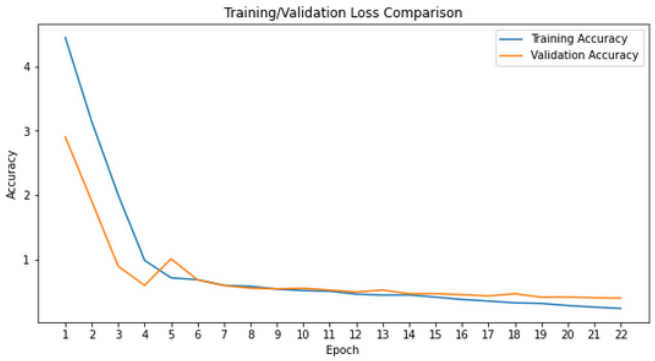

Figure 11. CNN Model Loss Comparison

When we look at the confusion matrix presented below we can understand that the probability of classifying correctly images that do not present a tumor is lower than the ones that present. A way to fix this issue would be to perform oversampling, yet it was not found that with this solution, the overall results, generally, were not as good as without the method being applied.

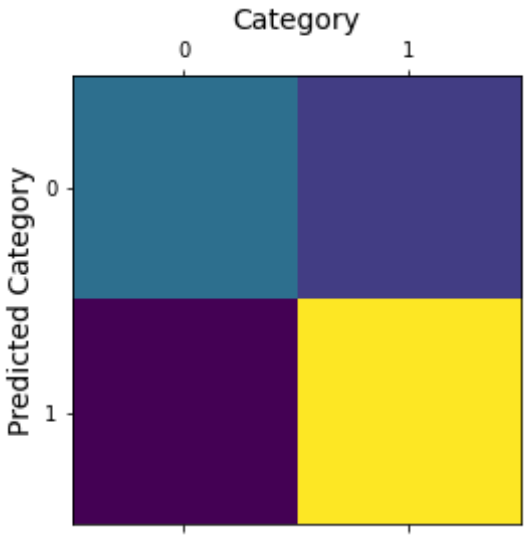

Figure 12. Confusion Matrix of the CNN Model

Table II
CNN MODEL STATISTICS

Train Accuracy	0.9007
Validation Accuracy	0.7255
Test Accuracy	0.8824
Train Loss	0.3005
Validation Loss	0.7022
Test Loss	0.4898
Precision	0.92
Recall	0.88
F1 Score	0.89

3) Transfer Learning Model Validation: In the images below we can see the evolution of the accuracy and the loss throughout epochs during the validation of the optimized transfer learning model. Given that the graph from training and the one from validation are relatively close to one another we can conclude that the model did not suffer from over-fitting and can generalize.

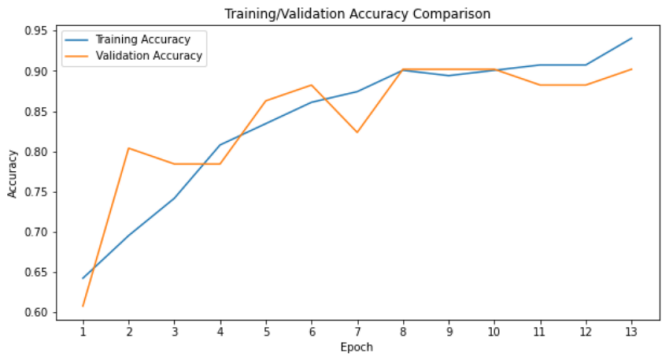

Figure 13. Transfer Learning Model Accuracy Comparison

In the table II we can observe more statistics about the model.

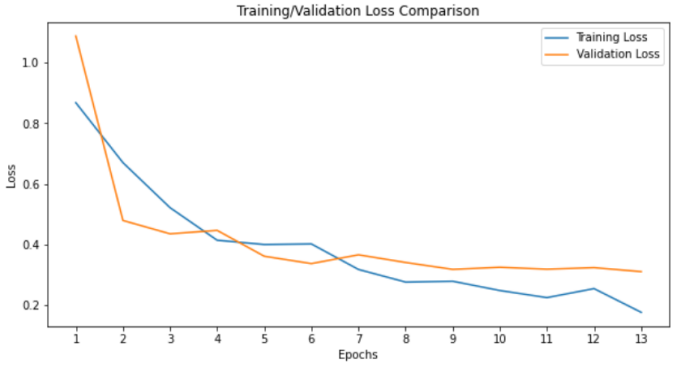

Figure 14. Transfer Learning Model Loss Comparison

From the confusion matrix below we can see that the model has a bigger probability of classifying an image as 1 (has tumor) than as a 0 (no tumor) and it also ends up classifying more 0s as 1s than the opposite. The major reason for this is the imbalance of classes in the data set but, as referred to at the beginning of the report, the attempts at fixing this by oversampling and undersampling ended up lowering the model performance as a whole.

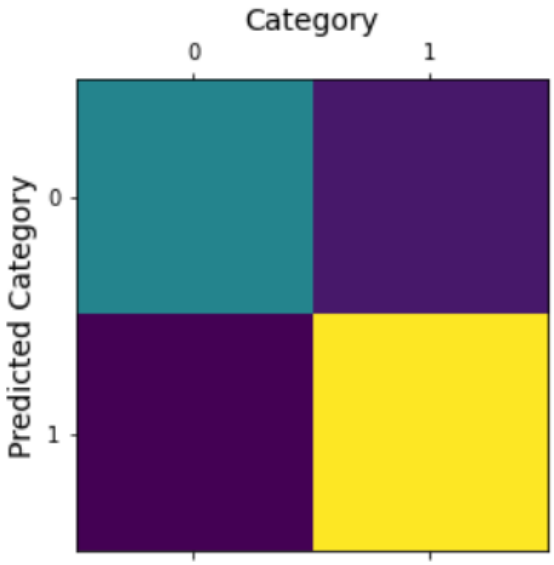

Figure 15. Confusion Matrix of the Transfer Learning Model

In the table III we can observe more statistics about the model.

VII. HYPER-PARAMETER SELECTION

A. SVM Model Hyper-Parameters

1) Grid Search: The way that we are going to search for the best hyperparameters is through GridSearchCV. GridSearchCV is a function that receives a list of values for the C parameter, the Gamma parameter, and the kernel and trains all combinations of parameters to find the best one. For each combination, the function performs 5-fold

Table III
TRANSFER LEARNING MODEL STATISTICS

Train Accuracy	0.9216
Validation Accuracy	0.9020
Test Accuracy	0.9216
Train Loss	0.2470
Validation Loss	0.3105
Test Loss	0.2470
Precision	0.93
Recall	0.92
F1 Score	0.92

cross-validation and iterates until convergence.

We entered as parameters for C the values 0.1, 1, 10, 100, for gamma the values 0.0001, 0.001, 0.1, 1 and kernel the options 'rbf','linear' and 'poly' and the combination with the best results was rbf with C=1 and gamma=0.0001.

B. Number of Epochs

The number of epochs is one of the most important hyper-parameters during the training phase of a deep learning model. It represents the number of times that the training data set will be seen by the model and therefore the number of times the model will try to adapt itself to the input data. If it is chosen incorrectly it can result in under-fitting which means that the model will not be able to correctly detect the relevant features or over-fitting where the model is unable to generalize beyond the training data. To select it, the process of validation was executed during 30 epochs.

1) Transfer Learning Model: According to the images below the best number of epochs in terms of validation accuracy is 13 and that is also one of the lowest values of validation loss. There 13 was the chosen value for the number of epochs.

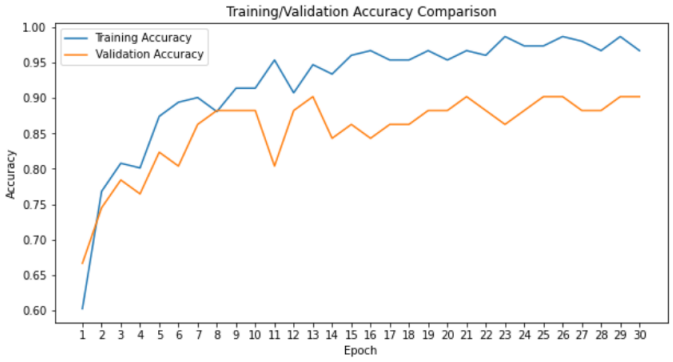

Figure 16. Transfer Learning Model Accuracy throughout the Epochs

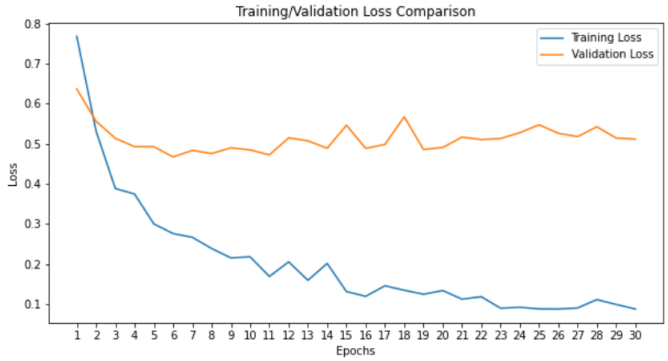

Figure 17. Transfer Learning Model Loss throughout the Epochs

C. Learning Rate

The learning rate is a hyper-parameter that controls how much the weights of the model change with the loss value throughout the epochs. According to this link [1], a common way to select it is by validating the model with different powers of 10 between 0.000001 and 1. Using this approach to our deep learning models we were able to select the one that allowed for the best performance.

1) CNN Model: As shown in the graphs below, when comparing all the tested learning rates, the ones who were the most consistent were the ones that present a 0.001 learning rate, yet the difference present in the losses from each learning rate was not significant, excluding the value 0.1 which ends up converging towards the other loss functions.

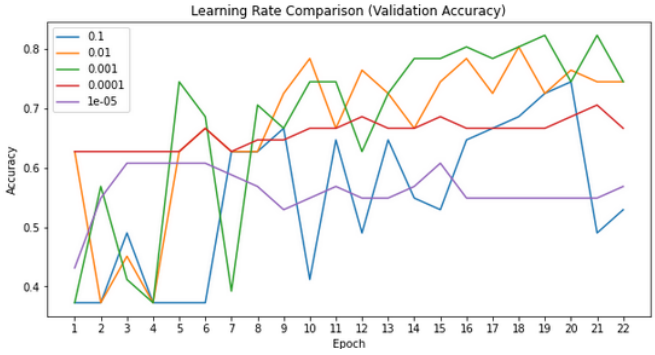

Figure 18. CNN Model Accuracy Comparison with different learning rates

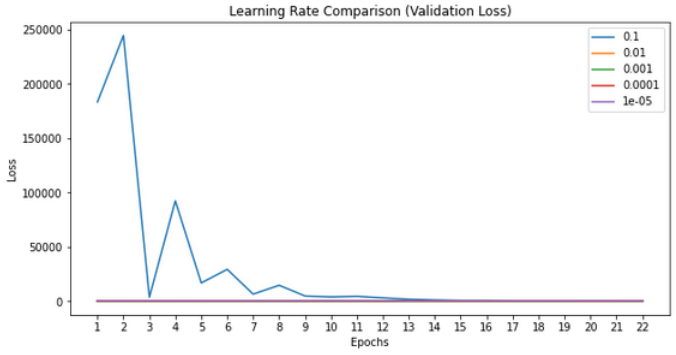

Figure 19. CNN Model Loss Comparison with different learning rates

2) Transfer Learning Model: According to the images below the best learning rate both in terms of validation accuracy and validation loss is 0.00001 and therefore it was chosen value for the learning rate of this model.

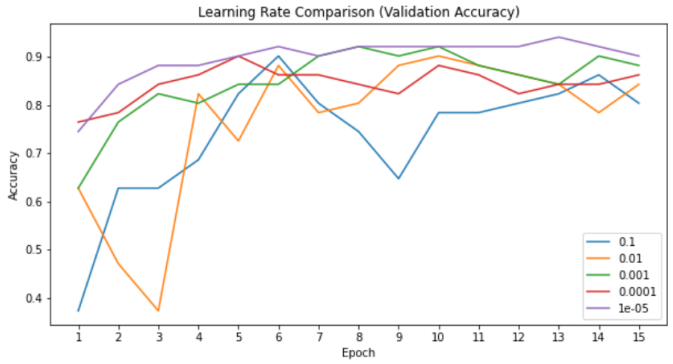

Figure 20. Transfer Learning Model Accuracy for different Learning Rates

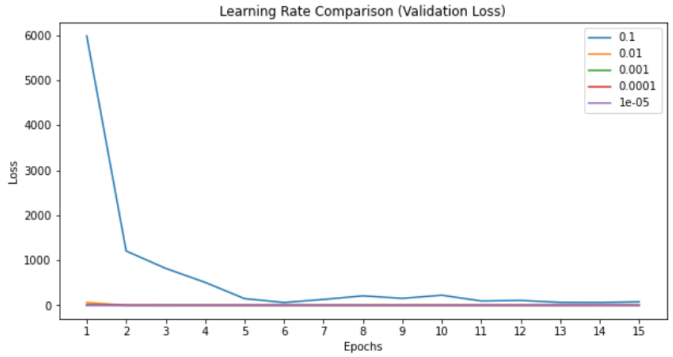

Figure 21. Transfer Learning Model Loss for different Learning Rates

VIII. Model Comparison and Previous Work

Among our models, the Transfer Learning Model has the best performance with 92% accuracy.

Comparing our models with already existing work by other authors our models were unable to achieve a better performance in any type of model. Although we used similar architectures we tried to create our version of data augmentation which was unable to surpass the performance of the one that the library Keras has. Additionally, we also tried to innovate with techniques of undersampling and oversampling which were not in any project analyzed by us but they did not provide better results.

IX. Conclusion

To conclude, this project contains the implementation of 3 models from 3 different types alongside the pre-processing of our chosen data set. Between all the models we obtained accuracies between 80% and 92%, with the Transfer Learning Model being the better performing one. Still, the results could be improved. For example, if we relied more on Keras to make our data augmentation we would be able to keep up with the previous work done on the dataset. Additionally, with more computation power we would be able to test increasing the image size or testing more complex models.

References

[1] Brownlee, J. (2019, August 6). How to Configure the Learning Rate When Training Deep Learning Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

[2] Brownlee, J. (2022). How to Configure Image Data Augmentation in Keras. Retrieved 4 May 2022, from https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[3] T. (2022, April 8). Brain Tumor Detection with Keras - 98% accuracy. Kaggle. https://www.kaggle.com/code/toobajamal/brain-tumor-detection-with-keras-98-accuracy.

[4] Team, D. (2021, May 19). Brain Tumor Classification using Machine Learning. DataFlair. https://data-flair.training/blogs/brain-tumor-classification-machine-learning/.

[5] Pal, T. (2022, February 5). Brain Tumor Detection Transfer Learning using VGG16. Medium. https://medium.com/mlearning-ai/brain-tumor-detection-transfer-learning-using-vgg16-a0d0a31f2e9.

[6] GeeksforGeeks. (2020, October 24). Intuition of Adam Optimizer. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

[7] Baheti, P. (2022, March 19). The Train, Validation, and Test Sets: How to Split Your Machine Learning Data. V7. https://www.v7labs.com/blog/train-validation-test-set.

[8] Brain MRI Images for Brain Tumor Detection. (2019, April 14). Kaggle. https://www.kaggle.com/datasets/navoneel/brain-mri-images-for-brain-tumor-detection/code.

[9] Gandhi, R. (2018, July 5). Support Vector Machine — Introduction to Machine Learning Algorithms. Medium. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[10] Shanmukh, V. (2022, January 1). Image Classification Using Machine Learning-Support Vector Machine(SVM). Medium. https://medium.com/analytics-vidhya/image-classification-using-machine-learning-support-vector-machine-svm-dc7a0ec92e01.

[11] B. (2021, April 26). Brain MRI Tumor Detection using SVM. Kaggle. https://www.kaggle.com/code/brendonim/brain-mri-tumor-detection-using-svm.

[12] Gandhi, A. (2021, May 20). Data Augmentation — How to use Deep Learning when you have Limited Data. AI Machine Learning Blog. https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[13] E. (2020, June 11). Brain Tumour Detection with CNN 96% accuracy. Kaggle. https://www.kaggle.com/code/ethernext/brain-tumour-detection-with-cnn-96-accuracy.