A black and white sign

Description automatically generated with low confidence

Software Engineering Department

Ort Braude College

Capstone Project Phase B

**Brain Tumor Radiogenomic Classification using CNN**

**22-1-R-5**

**Supervisor:**

Miri Weiss

**Students:**

Liron Shani, [liron.shani@e.braude.ac.il](mailto:liron.shani@e.braude.ac.il)

Daniel Polevikov, [daniel.polevikov@e.braude.ac.il](mailto:daniel.polevikov@e.braude.ac.il)

Table of Contents

[1. Introduction 3](#_Toc105685445)

[2. Background 3](#_Toc105685446)

[3. Project Review and Process Description 5](#_Toc105685447)

[3.1. CNN Architectures research 5](#_Toc105685448)

[3.2. Description of the research process 7](#_Toc105685449)

[3.3. Description of the testing process 24](#_Toc105685450)

[3.4. Discussion and Conclusions 24](#_Toc105685451)

[4. User Documentation 26](#_Toc105685452)

[4.1. Operating instructions - User’s Guide 26](#_Toc105685453)

[4.2. Maintenance Guide 27](#_Toc105685454)

[4.2.1. Description of software structure and runtime configuration using models 27](#_Toc105685455)

[4.2.2. Description of the organization and structure of the data 27](#_Toc105685456)

[4.2.3. Description of the operating environment of the system 28](#_Toc105685457)

[5. References 28](#_Toc105685458)

## **Introduction**

A malignant tumor in the brain is a life-threatening condition known as glioblastoma. It's both the most common form of brain cancer in adults and the one with the worst prognosis, with median survival being less than a year. The presence of a specific genetic sequence in the tumor known as MGMT promoter methylation has been shown to be a favorable prognostic factor and a strong predictor of responsiveness to chemotherapy. [1]

Currently, genetic analysis of cancer requires surgery to extract a tissue sample. Then it can take several weeks to determine the genetic characterization of the tumor. Depending upon the results and type of initial therapy chosen, a subsequent surgery may be necessary. If an accurate method to predict the genetics of the cancer through imaging (i.e., radiogenomics) alone could be developed, this would potentially minimize the number of surgeries and refine the type of therapy required. [1]

The Radiological Society of North America (RSNA) has teamed up with the Medical Image Computing and Computer Assisted Intervention Society (the MICCAI Society) to improve diagnosis and treatment planning for patients with glioblastoma.

In this project we predict if a patient has a malignant tumor in the brain based on an MRI scan of his brain. The challenge is to use machine learning techniques – Deep Learning -Convolution Neural Networks to make a prediction with the images and the metadata as input.

The methodology we used is developing a Convolutional neural network (CNN) in two architectures and testing various hyperparameters to optimize the trained system.

## **Background**

**CNN – Convolutional Neural Network**

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers. The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation and gradient descent, among others.

Chart

Description automatically generated

Fig 1: An overview of a CNN architecture and the training process [5]

Convolution Layers

Convolutional layers are the major building blocks used in convolutional neural networks.

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input result in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image. [6]

Pooling Layer

Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn, and the amount of computation performed in the network. The pooling layer summarizes the features present in a region of the feature map generated by a convolution layer. So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Table

Description automatically generated

Fig 2: Example of average pooling operation

Activation Function

An activation function is a function that is added into an artificial neural network in order to help the network learn complex patterns in the data. When comparing with a neuron-based model that is in our brains, the activation function is at the end deciding what is to be fired to the next neuron.

Diagram

Description automatically generated

Fig 3: Different Activation Functions and their Graphs

Fully Connected Layer

Fully Connected Layer is a layer where its neurons have full connection to all activations in the previous layer. Its purpose is to take the output of the convolution/pooling process (flattened to a vector) and use them to classify the image into a label.

Diagram

Description automatically generated

Fig 4: Fully Connected Layers

Softmax

The softmax step can be seen as a generalized logistic function that takes as input a vector of scores x ∈ Rn and outputs a vector of output probability p ∈ Rn through a softmax function at the end of the architecture. It is defined as follows:

Diagram

Description automatically generated

## **Project Review and Process Description**

We built a system that can classify whether a person has a brain malignant tumor given an MRI scan of his brain. We used a convolutional neural network (CNN) in various architectures and tested the system using different hyperparameters in order to find out what is the optimized way to build the network. We researched 2 CNN models: Densenet 121 and Inception V3, we tested each of them using the different hyperparameters. The hyperparameters we used for testing are batch size, epochs, dropout and learning rate.

## **CNN Architectures research**

**Densenet**

In this architecture each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. This is done to enable maximum information flow between the layers of the network. The features are combined by concatenating them. The L th layer has L inputs, consisting of the feature-maps of all preceding convolutional blocks. Its own feature-maps are passed on to all subsequent layers. There are L(L+1)/2 connections in an L-layer network.

Diagram, engineering drawing

Description automatically generated

Fig 5: A 5-layer dense block with a growth rate of k = 4. Each layer takes all preceding feature-maps as input [5].

DenseNet layers are very narrow, adding only a small set of feature-maps to the “collective knowledge” of the network and keep the remaining feature maps unchanged, and the final classifier decides based on all feature-maps in the network. A dense connectivity pattern requires fewer parameters, as there is no need to relearn unnecessary feature-maps. DenseNets have improved flow of information and gradients throughout the network, which makes the net easier to train. dense connections have a regularizing effect, which reduces overfitting on tasks with smaller training set sizes.

DenseNets Architecture:

* Dense connectivity, direct connections from any layer to all subsequent layers. The L th layer receives the feature-maps of all preceding layers, x0, . . ., xL−1, as input: xL = HL ([x0, x1, . . ., xL−1]), because of its dense connectivity we refer to this network architecture as Dense Convolutional Network (DenseNet).
* Composite function, define HL (·) as a composite function of three consecutive operations: BN, ReLU and a 3 × 3 Conv.
* Pooling layers, the concatenation operation used in dense connectivity cannot be performed if the size of feature-maps is different. An essential part of convolutional networks is down-sampling layers that change the size of feature-maps. To facilitate down-sampling in our architecture we divide the network into multiple densely connected dense blocks.

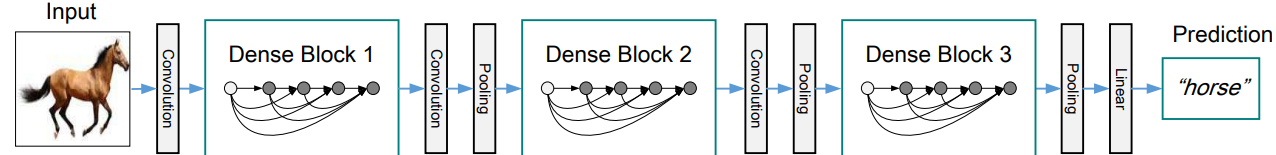


Fig 6: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling [5].

* Growth rate, if each function HL produces k feature maps, it follows that the L th layer has k0 +k ×(L−1) input feature-maps, where k0 is the number of channels in the input layer. The hyper parameter k is the growth rate of the network.
* Bottleneck layers, 1×1 convolution can be used as a bottleneck layer before each 3×3 convolution to reduce the number of input feature-maps and improve computational efficiency.
* Compression, to improve model compactness, we can reduce the number of feature-maps at transition layers by letting the following transition layer generate ⌊θm⌋ output feature maps, where 0 < θ ≤1.

**Inception V3**

The Inception V3 is a deep learning model based on Convolutional Neural Networks, which is used for image classification. The inception V3 is a superior version of the basic model Inception V1 which was introduced as GoogLeNet in 2014. As the name suggests it was developed by a team at Google. The inception v3 model was released in the year 2015, it has a total of 42 layers and a lower error rate than its predecessors. [2] Inception architecture achieved very good performance at relatively low computational cost. The main idea is to create a wider rather than deeper Network.

Diagram, arrow

Description automatically generated

Figure 7: Inception v3 architecture

There are no fully connected layers in this network, it helps to reduce a lot of parameters. The network basically builds by stacking these inception blocks on top of each other. Inception-v3 is a convolutional neural network architecture that builds on previous iterations of the Inception family by simplifying the architecture. The architecture of an Inception v3 network is that the output size of each module is the input size of the next one. The architecture uses variations of the reduction technique depicted and reduces the grid sizes between the Inception blocks whenever applicable progressively built.

## **Description of the research process**

We researched two different CNN models, with various learning rates, epochs, dropouts, and batch sizes on the same data set. We used Kaggle and Google colab that provided us services to run projects with big amount of data. We used brain MRI scans from Kaggle that are defined by three cohorts: Training, Validation, and Testing. These 3 cohorts are structured as follows: Each independent case has a dedicated folder identified by a five-digit number. Within each of these “case” folders, there are four sub-folders, each of them corresponding to each of the structural multi-parametric MRI (mpMRI) scans, in DICOM format. The exact mpMRI scans included are:

* Fluid Attenuated Inversion Recovery (FLAIR) - 53,688 scans in total
* T1-weighted pre-contrast (T1w) - 58,459 scans in total
* T1-weighted imaging with constract enhancement (T1wCE) - 71,019 scans in total
* T2-weighted (T2w) - 70, 723 scans in total

The hyperparameters evaluated are epochs (50,100,150), batch size (32, 64), learning rate (0.0005, 0.0000005) and dropout (0.2, 0.5).

The programming language that was used to write the system is Python. We used the following libraries for our system:

* pydicom, used to read the data (.dcm format).
* keras, used for the CNN models, layers, loss, and optimizer.
* os, used loading the data from the folders.
* sklearn, used for the train test data split.
* matplotlib , used to plot the accuracy and loss graphs.
* Pyqt5 , used for the graphic user interface

We defined the tested parameters this way:

* tp= True Positive, model prediction was there is a tumor, and the MRI scan was classified there is a tumor.
* tn = True Negative, model prediction was there is NO tumor, and the MRI scan was classified there is NO tumor.
* fp = False Positive, model prediction was there is a tumor, and the MRI scan was classified there is NO tumor.
* fn = False Negative, model prediction was there is NO tumor, and the MRI scan was classified there is a tumor.
* Accuracy = (tp + tn )/(tp + tn + fp + fn)

The relative part of the positive answer in all our parameters. The result will be between zero and one, with one being the best accuracy level and zero the worst.

* Precision = tp/(tp+ fp)

The number of positive class predictions that belong to the positive class.

* Recall = tp/(tp+ fn)

The number of correct positive predictions made from all positive predictions that could have been made.

The results we got are:

In the tables below there are the training (t\_loss, t\_accuracy), validation (val\_accuracy, val\_loss) and testing (recall, precision, accuracy) results. With those parameters we can see the results of each model in each phase. The tables indicate which hyperparameters impacted the most on the models and what is the type of data (filter) that gave us the best over all results. From the tables we found out that the best over all model is the Inception V3 with T2w filter and batch-size 32, 150 epochs, dropout 0.5 and learning-rate 0.0005. The second-best model is the Densenet 121 with the same filter and hyperparameters. Using the tables, we concluded that the hyperparameter which had the highest impact on our models is the learning rate (when it gets higher the accuracy rises). The same happens with the drop out, with the same learning rate we saw that in most of the models when the dropout gets higher the accuracy gets better. The cells that are marked in yellow are the cells with the best results in the column.

b = batch size, e = epochs, d = drop out, l = learning rate

**DenseNet 121**

The results of T1wCE filter on DenseNet 121 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **model** |
| 0.8771 | 0.8171 | 0.8280 | 0.8254 | 0.9569 | 0.9934 | 0.016 | b 64 e 100 d 0.5 l 0.0005 |
| 0.8417 | 0.8441 | 0.8310 | 0.8298 | 0.9705 | 0.9949 | 0.0158 | b 64 e 150 d 0.2 l 0.0005 |
| 0.8349 | 0.8041 | 0.8015 | 0.8029 | 0.7623 | 0.9817 | 0.0483 | b 32 e 50 d 0.5 l 0.0005 |
| 0.8308 | 0.8089 | 0.8031 | 0.8045 | 0.8828 | 0.9922 | 0.0214 | b 32 e 100 d 0.5 l 0.0005 |
| 0.8186 | 0.8618 | 0.8316 | 0.8333 | 0.9675 | 0.9977 | 0.0082 | b 64 e 150 d 0.5 l 0.0005 |
| 0.7909 | 0.8398 | 0.8061 | 0.8005 | 1.0817 | 0.9972 | 0.008 | b 32 e 150 d 0.5 l 0.0005 |
| 0.7868 | 0.8352 | 0.8015 | 0.7987 | 1.0832 | 0.9899 | 0.0281 | b 32 e 100 d 0.2 l 0.0005 |
| 0.7701 | 0.8678 | 0.8130 | 0.8145 | 1.0354 | 0.9962 | 0.0126 | b 32 e 150 d 0.2 l 0.0005 |
| 0.7580 | 0.8836 | 0.8159 | 0.8138 | 0.9233 | 0.9908 | 0.0286 | b 64 e 100 d 0.2 l 0.0005 |
| 0.7557 | 0.7479 | 0.7312 | 0.7339 | 0.9416 | 0.9807 | 0.0529 | b 64 e 50 d 0.5 l 0.0005 |
| 0.6785 | 0.6653 | 0.6430 | 0.6416 | 0.7201 | 0.8292 | 0.382 | b 32 e 150 d 0.2 l 0.0000005 |
| 0.6679 | 0.6190 | 0.5996 | 0.6015 | 0.6911 | 0.5949 | 0.9147 | b 64 e 150 d 0.5 l 0.0000005 |
| 0.6633 | 0.8999 | 0.7789 | 0.7789 | 1.0601 | 0.9811 | 0.0517 | b 32 e 50 d 0.2 l 0.0005 |
| 0.6615 | 0.5855 | 0.5654 | 0.5589 | 0.719 | 0.5352 | 1.0457 | b 64 e 50 d 0.5 l 0.0000005 |
| 0.6546 | 0.6312 | 0.6080 | 0.6069 | 0.6866 | 0.5839 | 0.936 | b 32 e 150 d 0.5 l 0.0000005 |
| 0.6536 | 0.6416 | 0.6168 | 0.6108 | 0.7106 | 0.7301 | 0.5616 | b 64 e 100 d 0.2 l 0.0000005 |
| 0.6529 | 0.5916 | 0.5702 | 0.5653 | 0.7167 | 0.5397 | 1.059 | b 32 e 50 d 0.5 l 0.0000005 |
| 0.6524 | 0.6517 | 0.6250 | 0.6316 | 0.6818 | 0.7228 | 0.5598 | b 32 e 100 d 0.2 l 0.0000005 |
| 0.6393 | 0.6084 | 0.5841 | 0.5745 | 0.7066 | 0.5679 | 0.9518 | b 64 e 100 d 0.5 l 0.0000005 |
| 0.6285 | 0.6724 | 0.6350 | 0.6355 | 0.7054 | 0.8262 | 0.393 | b 64 e 150 d 0.2 l 0.0000005 |
| 0.6152 | 0.6183 | 0.5882 | 0.5862 | 0.7012 | 0.5674 | 0.9843 | b 32 e 100 d 0.5 l 0.0000005 |
| 0.6126 | 0.6144 | 0.5843 | 0.5775 | 0.7161 | 0.6227 | 0.7174 | b 32 e 50 d 0.2 l 0.0000005 |
| 0.6119 | 0.9126 | 0.7594 | 0.7586 | 0.9754 | 0.9844 | 0.0434 | b 64 e 50 d 0.2 l 0.0005 |
| 0.5928 | 0.6144 | 0.5802 | 0.5712 | 0.7305 | 0.6295 | 0.7097 | b 64 e 50 d 0.2 l 0.0000005 |

The results of the filters on DenseNet 121 with the hyper parameters batch-size 32, 100 epochs, dropout 0.2 and learning-rate 0.0000005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.6259 | 0.6303 | 0.6134 | 0.6082 | 0.7294 | 0.6621 | 0.7521 | T1w |
| 0.6524 | 0.6517 | 0.6250 | 0.6316 | 0.6818 | 0.7228 | 0.5598 | T1wCE |
| 0.7489 | 0.6698 | 0.6398 | 0.6456 | 0.6938 | 0.6818 | 0.7029 | FLAIR |
| 0.7069 | 0.7091 | 0.6541 | 0.6599 | 0.6672 | 0.6802 | 0.7335 | T2w |

The results of the filters on DenseNet 121 with the hyper parameters batch-size 32, 150 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.8245 | 0.8436 | 0.8288 | 0.8215 | 1.0462 | 0.9967 | 0.0104 | T1w |
| 0.7909 | 0.8398 | 0.8061 | 0.8005 | 1.0817 | 0.9972 | 0.008 | T1wCE |
| 0.8929 | 0.8209 | 0.8247 | 0.8215 | 0.841 | 0.9956 | 0.0175 | FLAIR |
| 0.8627 | 0.9066 | 0.8659 | 0.8765 | 0.5334 | 0.9963 | 0.0111 | T2w |

The results of the filters on DenseNet 121 with the hyper parameters batch-size 64, 50 epochs, dropout 0.2 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.8062 | 0.7284 | 0.7422 | 0.7391 | 1.1309 | 0.9811 | 0.0508 | T1w |
| 0.6119 | 0.9126 | 0.7594 | 0.7586 | 0.9754 | 0.9844 | 0.0434 | T1wCE |
| 0.7804 | 0.8701 | 0.8048 | 0.8033 | 0.8352 | 0.985 | 0.0439 | FLAIR |
| 0.6964 | 0.9179 | 0.7830 | 0.7825 | 0.8873 | 0.9876 | 0.0359 | T2w |

The results of the filters on DenseNet 121 with the hyper parameters batch-size 64, 100 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.7307 | 0.8383 | 0.7860 | 0.7803 | 0.9908 | 0.9894 | 0.0296 | T1w |
| 0.8771 | 0.8171 | 0.8280 | 0.8254 | 0.9569 | 0.9934 | 0.016 | T1wCE |
| 0.9207 | 0.7641 | 0.7888 | 0.7916 | 1.2912 | 0.9925 | 0.0213 | FLAIR |
| 0.8830 | 0.8933 | 0.8680 | 0.8724 | 0.6671 | 0.9953 | 0.013 | T2w |

The results of the filters on DenseNet 121 with the hyper parameters batch-size 64, 150 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.8963 | 0.7954 | 0.8257 | 0.8228 | 1.0027 | 0.9903 | 0.0287 | T1w |
| 0.8186 | 0.8618 | 0.8316 | 0.8333 | 0.9675 | 0.9977 | 0.0082 | T1wCE |
| 0.8963 | 0.8412 | 0.8416 | 0.8408 | 1.0746 | 0.9998 | 7.43E-04 | FLAIR |
| 0.8440 | 0.9087 | 0.8572 | 0.8634 | 0.8287 | 0.9974 | 0.0085 | T2w |

In the graphs below we can see the behavior of the models with different combinations of the hyperparameters during the training and validation. They contain the accuracy and loss of the training and validation datasets in each epoch. In each table of graphs, the details of the model, its hyperparameters and filter are described. With the help of the graphs, we have concluded that our models are overfitting. In most of the graphs there are big gaps between the training and validation accuracy. In addition, we noticed that when the learning rate is higher (0.0005) the validation loss graph is less smooth and have more jumps and the value gets higher at some point. Despite the validation accuracy keeps rising this is the reason we conclude that our models are overfitting and instead of learning general feature the models learn features that are specific to the training data. In the validation loss graphs with the high learning rate have high spikes, we think that the reason that this happens is that the brain is a 3D object that is scan using the mri and divided into 2D slices. In the cases with the glioblastoma not all the slices contain the tumor but all the slices in the case are label as 1 (with glioblastoma). Combining the wrong labeling and the high learning rate our models might get high validation loss.

This table depicts a case of DenseNet architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0005 learning rate (batch-size = 32).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 32  learning rate = 0.0005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.5 |  |  |
| 100 | 0.2 |  |  |
| 150 | 0.2 |  |  |

This table depicts a case of DenseNet architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0000005 learning rate (batch-size = 32).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 32  learning rate = 0.0000005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.2 |  |  |
| 100 | 0.2 |  |  |
| 150 | 0.5 |  |  |

This table depicts a case of DenseNet architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0005 learning rate (batch-size = 64).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 64  learning rate = 0.0005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.5 |  |  |
| 100 | 0.2 |  |  |
| 150 | 0.5 |  |  |

This table depicts a case of DenseNet architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0000005 learning rate (batch-size = 64).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 64  learning rate = 0.0000005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.2 |  |  |
| 100 | 0.2 |  |  |
| 150 | 0.2 |  |  |

**Inception V3**

The results of T1wCE filter on Inception V3 are:

b = batch size, e = epochs, d = drop out, l = learning rate

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **model** |
| 0.6345 | 0.5831 | 0.5588 | 0.5553 | 0.7268 | 0.6563 | 0.6157 | b 32 e 100 d 0.2 l 0.0000005 |
| 0.7614 | 0.7892 | 0.7619 | 0.7608 | 0.9348 | 0.9723 | 0.0692 | b 32 e 100 d 0.2 l 0.0005 |
| 0.6053 | 0.5786 | 0.5499 | 0.5483 | 0.7232 | 0.5934 | 0.7082 | b 32 e 100 d 0.5 l 0.0000005 |
| 0.7850 | 0.7943 | 0.7746 | 0.7713 | 1.3434 | 0.972 | 0.0702 | b 32 e 100 d 0.5 l 0.0005 |
| 0.6130 | 0.5811 | 0.5535 | 0.5546 | 0.769 | 0.718 | 0.5429 | b 32 e 150 d 0.2 l 0.0000005 |
| 0.8229 | 0.8089 | 0.7999 | 0.7954 | 0.9574 | 0.9902 | 0.0282 | b 32 e 150 d 0.2 l 0.0005 |
| 0.6509 | 0.5804 | 0.5585 | 0.5555 | 0.7601 | 0.6646 | 0.6163 | b 32 e 150 d 0.5 l 0.0000005 |
| 0.7211 | 0.8413 | 0.7765 | 0.7783 | 0.9773 | 0.9859 | 0.0419 | b 32 e 150 d 0.5 l 0.0005 |
| 0.6455 | 0.5634 | 0.5396 | 0.5452 | 0.7113 | 0.578 | 0.6822 | b 32 e 50 d 0.2 l 0.0000005 |
| 0.7067 | 0.6661 | 0.6512 | 0.6499 | 0.7519 | 0.6899 | 0.5711 | b 32 e 50 d 0.2 l 0.0005 |
| 0.6213 | 0.5716 | 0.5452 | 0.5454 | 0.7107 | 0.5504 | 0.7642 | b 32 e 50 d 0.5 l 0.0000005 |
| 0.7651 | 0.7709 | 0.7510 | 0.7483 | 0.6419 | 0.8635 | 0.2843 | b 32 e 50 d 0.5 l 0.0005 |
| 0.6172 | 0.5759 | 0.5490 | 0.5448 | 0.731 | 0.6925 | 0.5784 | b 64 e 100 d 0.2 l 0.0000005 |
| 0.6324 | 0.5767 | 0.5520 | 0.549 | 0.7259 | 0.6976 | 0.5687 | b 64 e 100 d 0.2 l 0.0005 |
| 0.6276 | 0.5759 | 0.5505 | 0.556 | 0.7223 | 0.6242 | 0.6669 | b 64 e 100 d 0.5 l 0.0000005 |
| 0.8268 | 0.7191 | 0.7327 | 0.7364 | 1.594 | 0.9916 | 0.0251 | b 64 e 100 d 0.5 l 0.0005 |
| 0.5985 | 0.5850 | 0.5550 | 0.5534 | 0.7823 | 0.7725 | 0.4747 | b 64 e 150 d 0.2 l 0.0000005 |
| 0.5834 | 0.5836 | 0.5514 | 0.5539 | 0.7867 | 0.7766 | 0.4702 | b 64 e 150 d 0.2 l 0.0005 |
| 0.6264 | 0.5785 | 0.5529 | 0.5566 | 0.7605 | 0.692 | 0.5769 | b 64 e 150 d 0.5 l 0.0000005 |
| 0.8766 | 0.7144 | 0.7448 | 0.7429 | 1.7042 | 0.996 | 0.0122 | b 64 e 150 d 0.5 l 0.0005 |
| 0.6392 | 0.5688 | 0.5446 | 0.5375 | 0.7134 | 0.6043 | 0.6575 | b 64 e 50 d 0.2 l 0.0000005 |
| 0.6269 | 0.5701 | 0.5444 | 0.5375 | 0.7097 | 0.6066 | 0.6612 | b 64 e 50 d 0.2 l 0.0005 |
| 0.6240 | 0.5678 | 0.5416 | 0.5414 | 0.7161 | 0.5569 | 0.7515 | b 64 e 50 d 0.5 l 0.0000005 |
| 0.7304 | 0.7963 | 0.7541 | 0.756 | 0.9006 | 0.9626 | 0.093 | b 64 e 50 d 0.5 l 0.0005 |

The results of the filters on Inception V3 with the hyper parameters batch-size 32, 50 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.5755 | 0.6526 | 0.6188 | 0.6132 | 0.8523 | 0.7099 | 0.5353 | T1w |
| 0.7651 | 0.7708 | 0.7509 | 0.7483 | 0.6419 | 0.8635 | 0.2843 | T1wCE |
| 0.8208 | 0.7928 | 0.7713 | 0.7685 | 0.8461 | 0.9716 | 0.076 | FLAIR |
| 0.8872 | 0.8674 | 0.8526 | 0.8518 | 0.5319 | 0.98 | 0.051 | T2w |

The results of the filters on Inception V3 with the hyper parameters batch-size 32, 150 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.7987 | 0.7809 | 0.7782 | 0.778 | 1.0003 | 0.9744 | 0.0644 | T1w |
| 0.7211 | 0.8413 | 0.7765 | 0.7783 | 0.9773 | 0.9859 | 0.0419 | T1wCE |
| 0.5159 | 0.8277 | 0.6565 | 0.6511 | 1.8694 | 0.9946 | 0.0152 | FLAIR |
| 0.8892 | 0.9002 | 0.8758 | 0.8803 | 0.6258 | 0.9953 | 0.0145 | T2w |

The results of the filters on Inception V3 with the hyper parameters batch-size 32, 150 epochs, dropout 0.2 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.7252 | 0.7312 | 0.7176 | 0.7116 | 1.917 | 0.9951 | 0.014 | T1w |
| 0.8229 | 0.8089 | 0.7999 | 0.7954 | 0.9574 | 0.9902 | 0.0282 | T1wCE |
| 0.8205 | 0.8146 | 0.7874 | 0.7852 | 1.3503 | 0.9974 | 0.0097 | FLAIR |
| 0.8839 | 0.8699 | 0.8527 | 0.8579 | 0.7386 | 0.997 | 0.0099 | T2w |

The results of the filters on Inception V3 with the hyper parameters batch-size 64, 150 epochs, dropout 0.5 and learning-rate 0.0005 are:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **recall** | **precision** | **accuracy** | **val\_accuracy** | **val\_loss** | **t\_accuracy** | **t\_loss** | **filter** |
| 0.7888 | 0.8103 | 0.7935 | 0.7921 | 1.2438 | 0.995 | 0.0136 | T1w |
| 0.8766 | 0.7144 | 0.7448 | 0.7429 | 1.7042 | 0.996 | 0.0122 | T1wCE |
| 0.8579 | 0.7949 | 0.7890 | 0.7831 | 1.1993 | 0.9959 | 0.0109 | FLAIR |
| 0.8346 | 0.8818 | 0.8356 | 0.8414 | 0.7493 | 0.9968 | 0.0112 | T2w |

This table depicts a case of Inception V3 architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0005 learning rate (batch-size = 32).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 32  learning rate = 0.0005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.2 |  |  |
| 100 | 0.5 |  |  |
| 150 | 0.5 |  |  |

This table depicts a case of Inception V3 architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0000005 learning rate (batch-size = 32).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 32  learning rate = 0.0000005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.2 |  |  |
| 100 | 0.5 |  |  |
| 150 | 0.5 |  |  |

This table depicts a case of Inception V3 architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0005 learning rate (batch-size = 64).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 64  learning rate = 0.0005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.2 |  |  |
| 100 | 0.2 |  |  |
| 150 | 0.5 |  |  |

This table depicts a case of Inception V3 architecture by T1wCE filter, examining the influence of epochs and dropout on a 0.0000005 learning rate (batch-size = 64).

|  |  |  |  |
| --- | --- | --- | --- |
| T1wCE  Batch size = 64  learning rate = 0.0000005 | | | |
| epochs | dropout | Accuracy Plot | Loss Plot |
| 50 | 0.5 |  |  |
| 100 | 0.5 |  |  |
| 150 | 0.5 |  |  |

Some of the mri scans that were tested on inception v3 with the hyperparameters: batch size = 32, epochs = 150, dropout = 0.2 and learning rate = 0.0005 (the best model overall):

**Correct** predictions:

|  |  |
| --- | --- |
| Case 00000, filter T2w, image-240, labeled as 1 (with glioblastoma) | Case 00002, filter T2w, image-222, labeled as 1 (with glioblastoma) |
|  |  |
| Models' prediction: [5.2643187e-05, 9.9994731e-01] (there is glioblastoma) | Models' prediction: [7.9021083e-05, 9.9992096e-01] (there is glioblastoma) |

|  |  |
| --- | --- |
| Case 00003, filter T2w, image-260, labeled as 0 (no glioblastoma) | Case 00009, filter T2w, image-150, labeled as 0 (no glioblastoma) |
|  |  |
| Models' prediction: [0.99892527, 0.00107466] (there is no glioblastoma) | Models' prediction: [0.9976908, 0.00230919] (there is no glioblastoma) |

**Wrong** predictions:

|  |  |
| --- | --- |
| Case 00002, filter T2w, image-56, labeled as 1 (with glioblastoma) | Case 00002, filter T2w, image-57, labeled as 1 (with glioblastoma) |
|  |  |
| Models' prediction: [0.98132336, 0.01867671] (there is no glioblastoma) | Models' prediction: [0.69933987, 0.3006602] (there is no glioblastoma) |

**Correct** prediction but with **low possibilities**:

|  |  |
| --- | --- |
| Case 00002, filter T2w, image-58, labeled as 1 (with glioblastoma) | Case 00002, filter T2w, image-315, labeled as 1 (with glioblastoma) |
|  |  |
| Models' prediction: [0.48080605, 0.51919395] (there is glioblastoma) | Models' prediction: [0.40516225, 0.5948378] (there is glioblastoma) |

We created a simple Graphic User Interface (GUI) for our system. In the GUI, the user needs to upload an MRI scan of the brain. After that, the model makes a prediction and presents the result on the screen.

Here are examples of different MRI scans we have inserted into our system based **on Inception-V3 architecture with the parameters epochs = 150, learning rate = 0.0005, dropout = 0.5 and batch size = 32 trained on the Tw2 data**.

We chose this architecture with these parameters because we got the highest accuracy for this architecture and these parameters.

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated |  |

Main screen:

Graphical user interface, text, application, chat or text message

Description automatically generated

Loading Image:

A screenshot of a cell phone

Description automatically generated with low confidence

Example of a prediction with tumor found result:

A screenshot of a cell phone

Description automatically generated with low confidence

Example of a prediction with no tumor found result:

Graphical user interface, application, website

Description automatically generated

## **Description of the testing process**

One of the primary difficulties in any Machine Learning approach is to make the model generalized so that it is most accurate in predicting reasonable results with new data. Visualizing the training loss vs. validation loss or training accuracy vs. validation accuracy over several epochs is a good way to determine if the model has been sufficiently trained. This is important so that the model is not undertrained and not overtrained. If it is overtrained, it starts memorizing the training data which will, in turn, reduce its ability to predict accurately.

For that, we printed two graphs: accuracy and loss. The accuracy and loss data of the model for each epoch is stored in the history object. Accuracy method is used for measuring a classification model’s performance. The graph of the training loss vs. validation loss over the number of epochs will help the developer of the model to make informed decisions about the architectural choices that need to be made.

## **Discussion and Conclusions**

In our research of Two CNN architectures and different hyperparameters on the same data set, we concluded the effects of each hyperparameter on the results. The lowest training loss value was with the highest learning rate (0.0005) in both models. The highest accuracy in training was with the highest learning rate (0.0005) in both models. In the DenseNet 121 the lowest validation loss was with the lowest learning rate (0.0000005) while in the Inception v3 the lowest validation loss was with the lowest number of epochs (50). The highest validation accuracy value was with the highest learning rate (0.0005) in both models. The highest prediction accuracy was with the highest learning rate (0.0005) in both models. The highest precision accuracy was with the highest learning rate (0.0005) in both models. The highest recall accuracy was with the highest learning rate (0.0005) in both models. The hyperparameter which had the biggest impact in our research was the learning rate. In our research we got better results as the learning rate was higher. Despite getting better results with higher learning rate, we found out that with lower learning rate the network training was done more smoothly and with fewer jumps. but the end values are worse than with higher learning rate.

Example: Densenet 121 T1wCE Batch size = 32 dropout - 0.2 epochs =50

|  |  |
| --- | --- |
| learning rate = 0.0005 | learning rate = 0.0000005 |
|  |  |

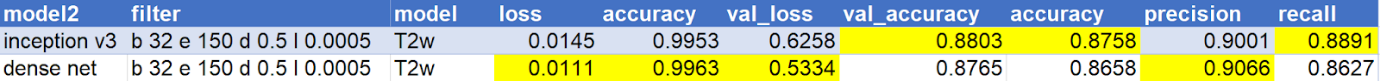
Another hyperparameter which had an impact in our research was the dropout, when it was higher (0.5 instead of 0.2) the validation accuracy of the model was higher. First, we noticed that when the dropout was higher (0.5 instead of 0.2) the validation accuracy of the model was higher.

Example: Inception T1wCE, Batch size = 64, learning rate = 0.0005, epochs = 50

|  |  |
| --- | --- |
| Dropout = 0.5 | Dropout = 0.2 |
|  |  |

Most of the significant improvement in accuracy and in loss was in the first 30 epochs. From 30 to 150 epochs, we still got an improvement but much smaller. Both networks perform almost the same, inception V3 got the highest validation accuracy = 0.8803 (denseNet validation accuracy = 0.8724) and the lowest validation loss = 0.5319 (densenet validation loss = 0.5334).

Our overall best model was the inception v3 with the hyperparameters: batch size = 32, epochs = 150, dropout = 0.2 and learning rate = 0.0005. Both models perform better on the T2w data.



Graphs of the best results for inception v3 and densenet 121:

|  |  |
| --- | --- |
|  |  |

Observing the graphs, we conclude that our models are overfitting. We came to this conclusion because of the gaps between the training and validation accuracy. Also, we noticed that in some models our validation loss gets worst, and this might happen because our models are no longer learning general features and instead, they are learning specific features of the training data. Some of the models with the high learning rate have very high validation loss in the training. This can happen because the brain is a 3D object that is scanned using the mri into 2D slices. In the cases that are labeled as 1 (with glioblastoma) some of the slices don’t contain a tumor. The combination of the miss labeling, and the high learning rate might lead to a high validation loss.

Throughout the research we coped with two challenges:

The first one was that the data we used from kaggle was images in DCM format. We had to make changes and convert the images pixel values to be in the range of 0-255. The second challenge was that we had to deal with a very big amount of data (over 250,000 images), something that is impossible to do on our computers. The solution we found was using the Google Colab platform, it also enabled us to work together on the same program.

## **User Documentation**

## **Operating instructions - User’s Guide**

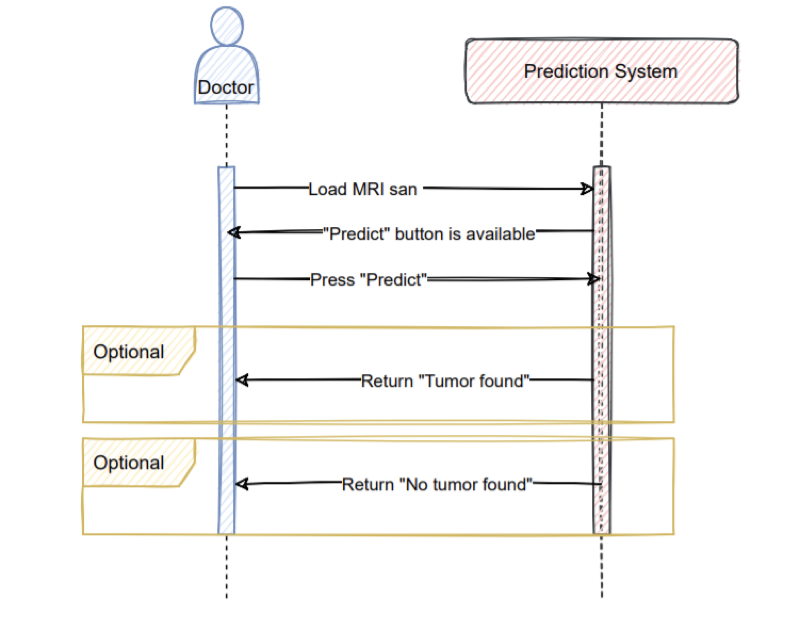
The purpose of our project is to research two different architectures with various hyperparameters and find the ideal architecture and hyperparameters, build a system that can correctly classify whether a person has a malignant tumor in the brain by his MRI scan. Our system is a program that creates and trains a CNN network with the hyperparameters we found are ideal for getting the most accurate results. The system prints graphs of accuracy and loss according to the network training process. We created the GUI of our system to show the user’s usage. In this interface, the user uploads an MRI scan, the system performs the classification and prints the result. We created the GUI with [PyQt5](https://pypi.org/project/PyQt5/) using Qt Designer and the code is written with Python.

We get from our training system a file that contains the model weights, and we load those weights in our CNN. The CNN is used to predict the MRI scan the user insert.

## **Maintenance Guide**

## **Description of software structure and runtime configuration using models**

Sequence diagram - using the system:

****

Program Run Process:

Diagram

Description automatically generated

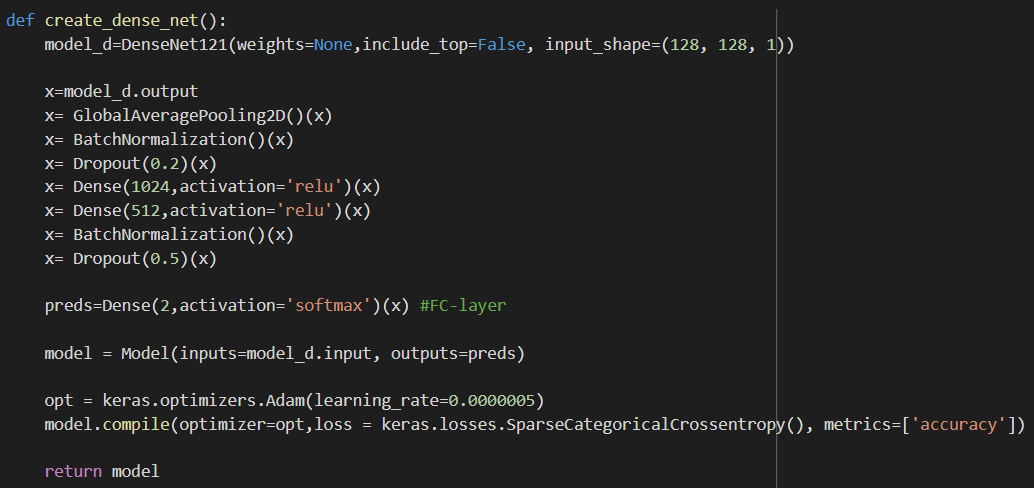
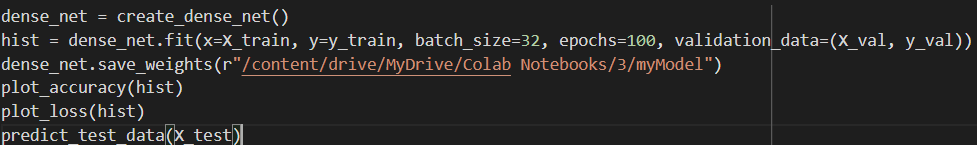
## **Description of the organization and structure of the data**

We used MRI brain scans from Kaggle. that are defined by three cohorts: Training, Validation, and Testing. These 3 cohorts are structured as follows: Each independent case has a dedicated folder identified by a five-digit number. Within each of these “case” folders, there are four sub-folders, each of them corresponding to each of the structural multi-parametric MRI (mpMRI) scans, in DICOM format. [3] The exact mpMRI scans included are:

* Fluid Attenuated Inversion Recovery (FLAIR) - 53,688 scans in total
* T1-weighted pre-contrast (T1w) - 58,459 scans in total
* T1-weighted imaging with constract enhancement (T1wCE) - 71,019 scans in total
* T2-weighted (T2w) - 70, 723 scans in total

## **Description of the operating environment of the system**

Train and test the model:

1. Go to the link [RSNA-MICCAI Brain Tumor Radiogenomic Classification | Kaggle](https://www.kaggle.com/competitions/rsna-miccai-brain-tumor-radiogenomic-classification/data) and download the training data.
2. Create a google account.
3. Upload the data to google drive.
4. Go to google colab.
5. Upload the file densenet.ipynb or inception v3.ipynb.
6. Update the path to the labels ****
7. Update the path to the data and specify the filter type****
8. Choose your model hyperparameters (dropout and learning rate)****
9. Choose the batch size and epochs, change the path of the saved weights****
10. Press runtime and then run all

Run the gui:

1. open and run the Tumor\_predict.py
2. load dcm format mri scan using the load button
3. press predict

## **References**

[1] RSNA-MICCAI Brain Tumor Radiogenomic Classification.

[2] Inception V3 Model Architecture by [Adith Narein T](https://iq.opengenus.org/author/adith/)

[3] https://www.kaggle.com/competitions/rsna-miccai-brain-tumor-radiogenomic-classification/data

[4] Mohsen, H., El-Dahshan, E.A., El-Horbaty, E.M., Salem, A.M. Brain tumor type classification based on support vector machine in magnetic resonance images, Annals Of “Dunarea De Jos” University of Galati, Mathematics, Physics, Theoretical mechanics, Fascicle II, Year IX (XL), No. 1; 2017.

[5] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do and Kaori Togashi, “Convolutional neural networks: an overview and application in radiology”, Insights into Imaging volume 9, pages 611–629 (2018).

[6] [Jason Brownlee](https://machinelearningmastery.com/author/jasonb/), " How Do Convolutional Layers Work in Deep Learning Neural Networks?", April 17, 2019 in [Deep Learning for Computer Vision](https://machinelearningmastery.com/category/deep-learning-for-computer-vision/) Tweet.