
**DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING**

**EE491 PROJECT REPORT**

# Binary Classification of Tumors from MR Images using Deep Learning

UTKU ACAR 250206062

MEHMET ZÜBEYİR ÜNLÜ

DATE: 15/01/2022

# ABSTRACT

In this project report there are both theoretical knowledge about and practical applications about Binary Classification of Brain Tumors with using Convolutional Neural Networks by Deep Learning. There is also introduction about each layer in CNNs. The aim of this project is designing a deep learning model-based decision support system for brain tumor classification with the highest accuracy possible to help physicians. Since the class of tumors in the brain is critical to choosing the treatment method, this model can aid in this critical decision process of physicians. Since algorithms for deep learning are complex, functions in Keras library have been used to create model. The model created on Python programming language. There are many cases to reach best accuracy in this report. In the end, the project reached its design goal.

# TABLE OF CONTENTS

**ABBREVIATIONS** ..... 3
**LIST OF FIGURES** ..... 4
**LIST OF TABLES** ..... 5
**1. INTRODUCTION** ..... 6
**1.1. What is Deep Learning and Why we are using it?** ..... 6
**1.2. Binary Classification** ..... 6
**1.3. Difference Between Low- and High-Grade Gliomas** ..... 7
**1.4. General Working Scheme of the Project** ..... 8
**2. PROBLEM DEFINITON** ..... 9
**2.1. Convolutional Neural Networks** ..... 9
**2.2. Perceptron** ..... 9
**2.3. Convolutional (Filtering) Layer** ..... 10
**2.4. Initialization of Filter Weights** ..... 11
**2.5. Updating of Filter Weights** ..... 11
**2.6. Max Pooling Layer** ..... 12
**2.7. Flattening Layer** ..... 13
**2.8. Dropout Layer** ..... 13
**2.9. Dense Layer (Multilayer Perceptron, Fully Connected Layer)** ..... 14
**3. PROPOSED SOLUTION** ..... 15
**3.1. Model Creating** ..... 15
**3.2. Training a Model** ..... 15
**3.3. Optimization and Early Stopping Algorithms** ..... 16
**4. RESULTS AND DISCUSSIONS** ..... 17
**5. CONCLUSIONS** ..... 19
**REFERENCES** ..... 20

# ABBREVIATIONS

**HGG :** High Grade Glioma

**LGG :** Low Grade Glioma

**CNN :** Convolutional Neural Network

**ReLU :** Rectified Linear Unit

# LIST OF FIGURES

**Figure 1** HGG vs LGG Example [2] ..... 7
**Figure 2** CNN Example [3] ..... 9
**Figure 3** Perceptron [4] ..... 10
**Figure 4** Convolutional Filtering Example [5] ..... 10
**Figure 5** Xavier Uniform Initializer [6] ..... 11
**Figure 6** Updating Filter weights and bias [7] ..... 11
**Figure 7** Calculations of Gradients [8] ..... 12
**Figure 8** Detailed Formula for Filter Weight Update [13] ..... 12
**Figure 9** Max Pooling Example [9] ..... 13
**Figure 10** Flattening Layer Example [10] ..... 13
**Figure 11** Dropout Layer Example [11] ..... 14
**Figure 12** Dense Layer Example [12] ..... 14

# LIST OF TABLES

**Table 1** Accuracy and Loss Results for Several Cases ..... 17

# 1. INTRODUCTION

Since the technology evolved so much such that, they can recognize any objective from any image even from noisy environments, machines can also learn from before and predict future with using old data. This learning process is heavily dependent on finding “Feature” to distinguish any object or person from each other. The name of this distinction of any object with using these features is “Machine Learning”. There are many ways to train a machine just like “Supervised Learning” where data are labelled or “Unsupervised Learning” where data is not labelled or “Reinforcement Learning” which based on reward/penalty system. After some time, the researchers think that what if we have a model that we do not need to give features and force the model to learn the distinctive features by itself: The self-learning models are named as “Deep Learning”. These kinds of models are more detailed subdivision of machine learning. This project is aims to use this advantage of self-learning ability of deep learning models to classify brain tumors as HGG or LGG and aid physicians or radiology technicians with this classification process.

## 1.1. What is Deep Learning and Why we are using it?

Deep learning is a subfield of machine learning which focuses on mimicking neuron activities such as deciding, learning, recognizing in our brains [1]. Deep learning models are used to find complex relationships between variables even if they look like independent.

## 1.2. Binary Classification

Binary Classification is a labelling method consists of only 2 different classes. Used in;

- Medical industry to determine patient has specific disease or not.
- Control industry to check if the corresponding product satisfies specifications or not.

Despite we have only 2 outputs (0,1) we have 4 cases due to output accuracy:

- False Negative
- False Positive
- True Negative
- True Positive

## 1.3. Difference Between Low- and High-Grade Gliomas

There are 2 grades of Gliomas. These grades based on differentiation of cells, levels of aggression, etc.

- Low-Grade Gliomas (LGG) are less aggressive and has slower growth rate.
- High-Grade Gliomas (HGG)are more aggressive and has rapid growth rate.

The example can be seen in Figure 1.

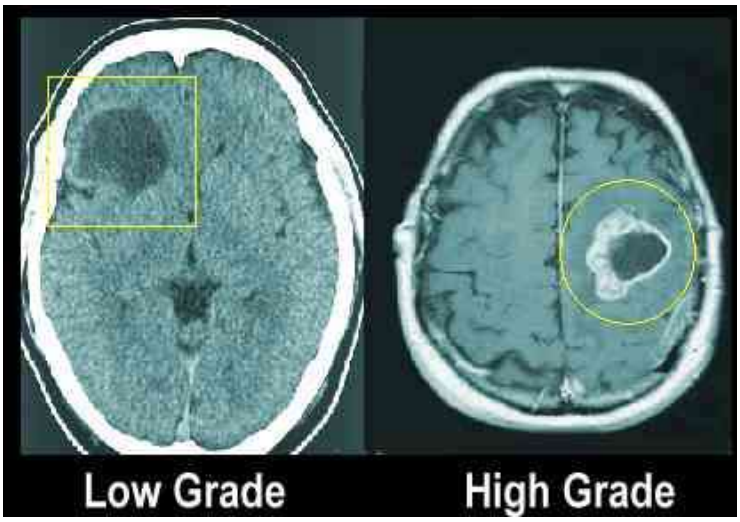


**Figure 1** HGG vs LGG Example [2]

## 1.4. General Working Scheme of the Project

General Scheme of this project has been given below:

1. Input (Brain MR Images)
2. Zero Padding (Kernel Fitting)
3. Convolution Filtering (Getting Features)
4. Max-Pool (Removing Unnecessary Features)
5. Convolution Filtering (Getting More Detailed Features)
6. Max-Pool (Removing Unnecessary Detailed Features)
7. Flattening (Matrix to Array Conversion)
8. Multi-Layer Perceptron (Dense) (Comparing multiple samples)
9. Activation Function (Decide HGG or LGG)
10. Output (Giving Prediction Results)

# 2. PROBLEM DEFINITON

Problem is, understanding how CNNs work by figuring out which layer is responsible for which operations exactly. There are couple of layers and concepts to explain below.

## 2.1. Convolutional Neural Networks

Convolutional Neural Networks are regularized versions of Multilayer Perceptron or Fully Connected Layers. There are additional operations in CNNs for regulations such as convolution, pooling, flattening layers, etc. These layers help the model to find distinctive features in given images. An example of the CNN structure can be seen in Figure 2.

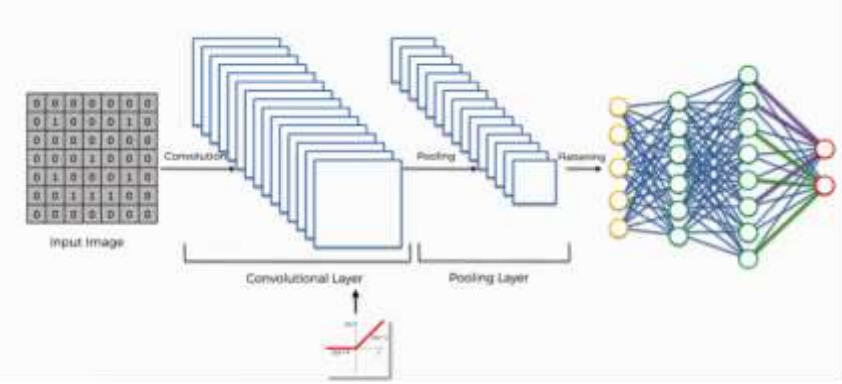


**Figure 2** CNN Example [3]

## 2.2. Perceptron

Perceptron is a fundamental unit of Neural Networks, and it is the equivalent of neurons in our neural system. Perceptron gets inputs and multiply these inputs with their weight coefficients and sums up all these multiplication results, then uses the activation function to produce an output. Since they are very useful for classifying data, perceptron has also a drawback, and it is that “they can only classify linearly separable sets of

vectors.” But this drawback can be eliminated by adding non-linear activation functions such as “ReLU” or “Sigmoid”. The structure of a perceptron can be seen in Figure 3.

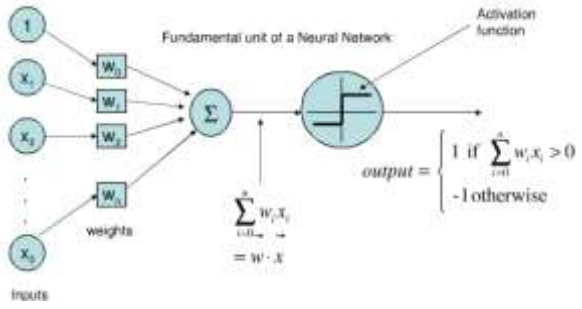


**Figure 3** Perceptron [4]

## 2.3. Convolutional (Filtering) Layer

Convolutional Layer is the layer that uses various sizes of 2 or 3 dimensional kernels to extract features from given image by applying convolution (Cross Correlation) operation using corresponding parts of image and filter kernel weights. There is a need for zero padding before convolutional filtering to avoid data loss at edges of the image.

The name “convolution” is usually used for “cross-correlation” since both are the same operation except in convolution there are reversing operations before sliding the kernel around an image to extract features. Since convolution kernels are symmetric, reversing a kernel by 180 degrees results same kernel as before. The convolutional layer operations can be seen in Figure 4.

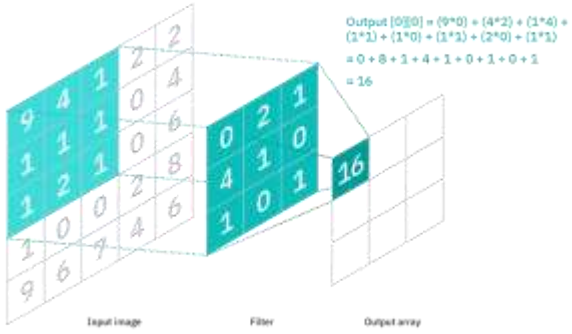


**Figure 4** Convolutional Filtering Example [5]

## 2.4. Initialization of Filter Weights

Since we want to learn the correct weights by backpropagation and not giving exact weights to each filter, we should initialize the filter weights somehow. I have used Xavier uniform initializer, which is the default initializer in Keras library to initialize filter weights. The range formula for Xavier uniform initializer is shown below in Figure 5.

$$\pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$$

$n_i$ : Number of Input Connections

$n_{i+1}$ : Number of Output Connections

**Figure 5** Xavier Uniform Initializer [6]

## 2.5. Updating of Filter Weights

Filter weights are updated to minimize loss function output by using backpropagation. This is important for the self-learning process. Binary cross-entropy used as a loss function in the project model. Figure 6 shows the gradient descent algorithm which is used to optimize models with updating weight coefficients. If the learning rate "a" is too high, there could be an overshooting problem, and if a is too low then learning time would increase drastically. So, picking a suitable learning rate is very important for optimization.

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$b = b - \alpha \frac{\partial L}{\partial b}$$

**(L - Loss function)**
**(W - Filter Weight)**
**(b - Bias)**
(α – *Learning Rate*)

**Figure 6** Updating Filter weights and bias [7]

Filter weights are updated to reduce loss while increasing accuracy. So, gradient of loss function due to input and weight coefficients. Calculations of these gradients can be seen in Figure 7.

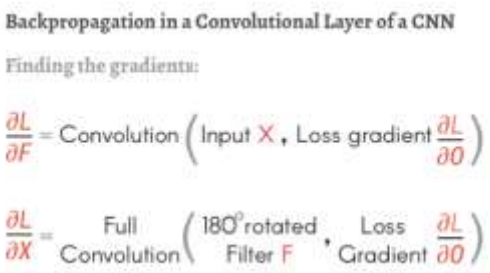


**Figure 7** Calculations of Gradients [8]

The detailed calculations of gradients can be seen in Figure 8.

$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2} \delta^l_{i,j} o^{l-1}_{i+m',j+n'} \quad (13)$$

$$= \mathrm{rot}_{180^\circ}\left\{\delta^l_{i,j}\right\} * o^{l-1}_{m',n'} \quad (14)$$

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} w^{l+1}_{m,n} f'\left(x^l_{i',j'}\right)$$

$$= \mathrm{rot}_{180^\circ}\left\{\sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'+m,j'+n} w^{l+1}_{m,n}\right\} f'\left(x^l_{i',j'}\right) \quad (21)$$

$$= \delta^{l+1}_{i',j'} * \mathrm{rot}_{180^\circ}\left\{w^{l+1}_{m,n}\right\} f'\left(x^l_{i',j'}\right) \quad (22)$$

$E$ - Error Function

$w^l_{m',n'}$ - Convolutional Filter Weights at layer l

$\delta^l_{i,j}$ - Gradient of $\frac{\partial E}{\partial w^l_{m',n'}}$ at layer l

$o^{l-1}_{m',n'}$ -Output vector at previous layer l-1

$x^l_{i',j'}$ - Single pixel from input feature map at layer l

$f'(.)$ - Derivative of activation function

**Figure 8** Detailed Formula for Filter Weight Update [13]

## 2.6. Max Pooling Layer

Max pooling is used to reduce dimensionality. This results in less required computation power for learning the features. Max pooling algorithm takes input data and chooses the maximum value of the given input. In this project input is 2D Matrix parts which are results of convolutional filtering and max pooling layer has size of 2 (2x2 kernel). An example of a max pooling operation can be seen in Figure 9.

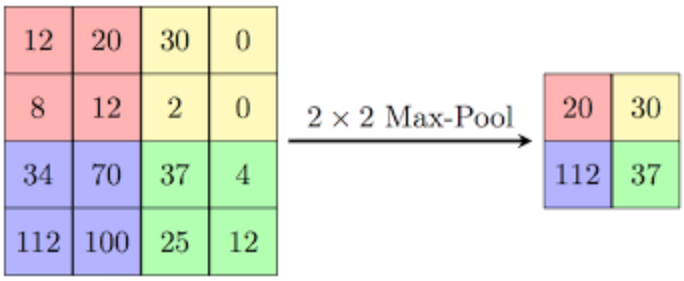


**Figure 9** Max Pooling Example [9]

## 2.7. Flattening Layer

Flattening Layer is responsible for converting N x N input (N-Dimensional) to 1x N*N output (1-Dimensional) to make the data processable by multilayer perceptron (Dense). An example of a flattening layer can be seen in Figure 10.

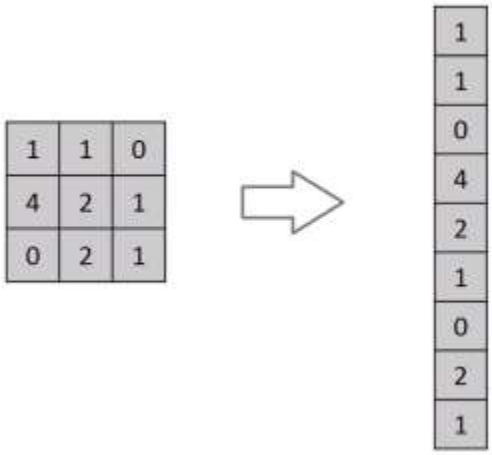


**Figure 10** Flattening Layer Example [10]

## 2.8. Dropout Layer

Dropout Layer is used to prevent overfitting. This layer resets (makes 0) random input due to frequency as given rate and multiplies other ones which have not been reset with 1/(1-rate) to maintain summation of all inputs same. Dropped (turned off) neurons (perceptrons) can be seen in red and active ones can be seen in green in Figure 11.

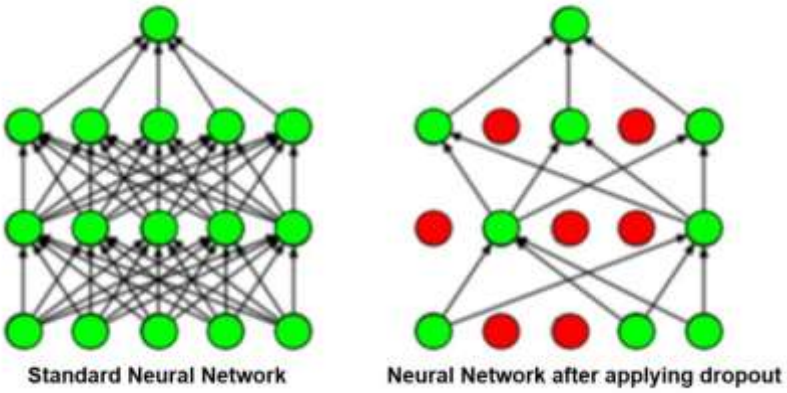


**Figure 11** Dropout Layer Example [11]

## 2.9. Dense Layer (Multilayer Perceptron, Fully Connected Layer)

Dense layer consists of perceptrons. Each perceptron from each layer has a connection between all perceptrons at the next layer. The name “Fully Connected Layer” comes from this topology. In this project, dense layers are used to predict tumor grade for the given Brain MRI (HGG or LGG) by processing images with calculated features which are found by convolutional layers before. The dense layer structure can be seen in Figure 12.

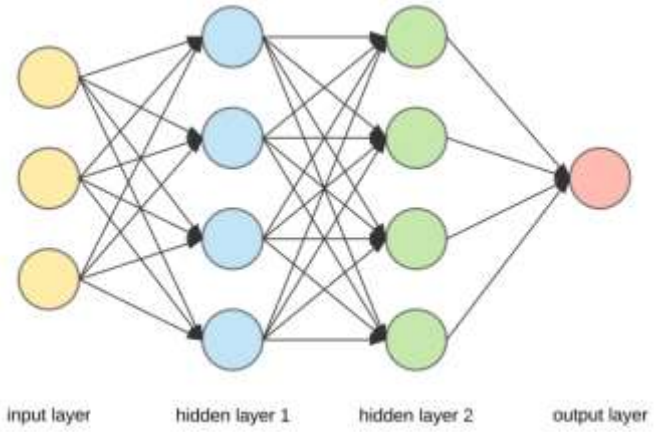


**Figure 12** Dense Layer Example [12]

# 3. PROPOSED SOLUTION

Creating a CNN based deep learning model to predict classes of tumors accurately as possible.

## 3.1. Model Creating

The CNN-based model should be created to predict with high accuracy. The Project's model has 4 convolutional layers, details given below:

Output size of filters is given by 16,32,64,128. Kernel size of convolutional filters is 3 by 3.

There are two kind of activation function in this model and there are Rectified Linear Unit (ReLU) mostly except last dense layer. Sigmoid is used at last layer since this project aims to classify tumors by binary classification and sigmoid used in binary classification because it has better results than ReLU even it is more costly than ReLU since ReLU is just taking maximum of value between input and 0 while output of sigmoid is bounded to exponential calculations.

Input shape for just first layer has been given as 240 by 240 with 3 channels.

Each Convolutional layer followed by 2x2 Max Pooling Layer.

There are Dropout Layer at just before and between dense layers with rate of 0.25.

There are 3 dense layers with sizes 64,16,1. The last dense layer has a size of 1 and the output of it is in the range of [0,1] so if the output is 0 then the input image is labeled as HGG, and if the output is 1 then it is labeled as LGG.

## 3.2. Training a Model

Models should be trained on reliable and consistent datasets. This project's model is trained on a reliable dataset named as "Brats2020" which stands for "Brain Tumor Segmentation Challenge".

This dataset was prepared by Perelman School of Medicine, which is a subdivision of University of Pennsylvania. The dataset consists of MR images of 369 different patients; each had 155 axial slices. Generally, the $78^{th}$ slices are the most detailed slice. Since deep learning models require more than 369 slices to process, and the dataset has a limited number of patients, more than one slice has been used for each person. Slices are centered about $78^{th}$ slices.

To be exact, slices have been taken from 74 to 83 of patients numbered less than or equal to 225; so, there are 2250 slices for HGG, and slices from 64 to 93 of patients numbered between 260 and 336; so, there are 2250 slices for LGG, too. Then some of the slices are removed since they are like each other At the end number of slices for HGG and LGG for project model is reduced to 1127.

## 3.3. Optimization and Early Stopping Algorithms

Various parameters have been tried to optimize the project's model by increasing accuracy while decreasing loss at the same time. Dropout rate, epoch number, steps per epoch, validation steps, patience for early stopping have been changed and extra dense layers have been added to optimize the model. Also, Early Stopping and Model Checkpoint functions have been used which are in Keras library.

Early Stopping compares validation accuracy with previous validation accuracy values to find if validation accuracy increased by minimum delta or not. The number of previous validation accuracy values is based on the "Patience" parameter in Early Stopping function.

Model Checkpoint function has been used to store best model which has best validation accuracy up to the epoch at that time.

# 4. RESULTS AND DISCUSSIONS

Best accuracy has been found when Epoch was 50 but early stopped at 34 while steps per epoch was 99, validation steps was 16, minimum delta was 0.001 which was same for all cases and patience was 10. This configuration reached 99.7% accuracy and 1.7% loss. For determining false negative and false positive cases, the accuracy broke into two as HGG and LGG accuracy. For the best case the model predicts LGG patients by 100% and HGG patients by 99.4% correctness and predict HGG falsely by 0.6%. Generally, while we are increasing steps per epoch, the accuracy is also increasing. These results can be found in Table 1.

| Epoch | Steps per epoch | Validation Steps | Minimum Delta (Early Stopping) | Patience (Early Stopping) | Accuracy General (Test Data) | HGG accuracy (Test Data) | LGG Accuracy (Test Data) | Loss (Test Data) |
|---|---|---|---|---|---|---|---|---|
| 34/50 | 99 | 16 | 0.001 | 10 | 0.9970 | 0.9940 | 1.000 | 0.0172 |
| 30 | 70 | 16 | 0.001 | 10 | 0.9881 | 0.9940 | 0.9821 | 0.0668 |
| 30 | 50 | 20 | 0.001 | 6 | 0.9643 | 0.9583 | 0.9702 | 0.0845 |
| 30 | 50 | 16 | 0.001 | 10 | 0.9642 | 0.9523 | 0.9761 | 0.0901 |
| 30 | 60 | 16 | 0.001 | 6 | 0.9583 | 0.9404 | 0.9761 | 0.1660 |
| 30 | 60 | 16 | 0.001 | 10 | 0.9554 | 0.9107 | 1.0000 | 0.0814 |
| 30 | 30 | 16 | 0.001 | 10 | 0.9286 | 0.8750 | 0.9821 | 0.1842 |

**Table 1** Accuracy and Loss Results for Several Cases

I have doubt that the 99.7% accuracy is too good to be true but since we are forcing the model to run this data for 34 times, it finds the true label at the end, and doing so there is no overfitting thanks to the dropout layer and early stopping algorithm. This high accuracy value could be resulted by the process of taking multiple slices for each patient because the probability of being both a training and validation set of the image is not 0.

I think that datasets with more patient results are more reliable and consistent.

This project can be improved by classifying the brain tumor type more diversely by 4 classes (for each grade (1,2,3,4)) instead of 2 and segmenting the existing tumor from the MRI.

Since this model is not specific to just brain MRI's, the model can be used to classify any object from images after a new training operation.

# 5. CONCLUSIONS

To conclude the project, I have created a model with 99.7 percent accuracy and 1 percent loss, which satisfies my design target. I gained a lot of experience in dataset manipulation, creating models to accurately classify tumors, and fundamental concepts of Convolutional Neural Networks and Deep Learning in Python. Also, I have seen how accuracy and loss differ with changing various parameters and changing layer configurations.

# REFERENCES


**[1]** J. Brownlee, “What is deep learning?,” Machine Learning Mastery, 14-Aug-2020. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/. [Accessed: 06-Jan-2022].

**[2]** “MRI Appearance of Primary Brian Tumors,” MRI picture of a glioblastoma. [Online]. Available: http://www.aboutcancer.com/mri_gbm.htm. [Accessed: 08-Jan-2022].

**[3]** C. Barrows, “Multiclass classification with cnn-2d,” Multiclass Classification with CNN-2D. [Online]. Available: https://morioh.com/p/ac7c0c4430b8. [Accessed: 08-Jan-2022].

**[4]** “Perceptron (algılayıcı) ve öğrenme Kuralı,” Çevrimiçi Fen Eğitimi, 12-May-2015. [Online]. Available: https://ahmetasimsengul.wordpress.com/2015/05/12/perceptron-algilayici-ve-ogrenme-kurali/. [Accessed: 08-Jan-2022].

**[5]** Texasdave, “Image classification tutorial with Mnist Fashion,” Kaggle, 11-Jul-2021. [Online]. Available: https://www.kaggle.com/texasdave/image-classification-tutorial-with-mnist-fashion. [Accessed: 08-Jan-2022].

**[6]** J. Dellinger, “Weight initialization in neural networks: A journey from the basics to kaiming,” Medium, 04-Apr-2019. [Online]. Available: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79. [Accessed: 08-Jan-2022].

**[7]** B. Kanani, “Setting dynamic learning rate while training the neural network,” Machine Learning Tutorials, 09-Dec-2019. [Online]. Available: https://studymachinelearning.com/setting-dynamic-learning-rate-while-training-the-neural-network/. [Accessed: 08-Jan-2022].

**[8]** P. Solai, “Convolutions and backpropagations,” Medium, 18-Apr-2018. [Online]. Available: https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c. [Accessed: 08-Jan-2022].

**[9]** “Max-pooling / pooling,” Max-pooling / Pooling - Computer Science Wiki. [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling. [Accessed: 08-Jan-2022].

**[10]** T. Ergin, “Convolutional Neural Network (convnet yada CNN) Nedir, Nasıl çalışır?,” Medium, 22-Feb-2020. [Online]. Available: https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad. [Accessed: 08-Jan-2022].

**[11]** S. Zivkovic, “#018 pytorch - popular techniques to prevent the overfitting in a neural networks,” Master Data Science, 08-Nov-2021. [Online]. Available: https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/. [Accessed: 08-Jan-2022].

**[12]** S. Hussain, “Building a convolutional neural network: Male vs female,” Medium, 06-Apr-2020. [Online]. Available: https://towardsdatascience.com/building-a-convolutional-neural-network-male-vs-female-50347e2fa88b. [Accessed: 08-Jan-2022].

**[13]** Jefkine, “Backpropagation in Convolutional Neural Networks,” DeepGrid, 05-Sep-2016. [Online]. Available: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/. [Accessed: 06-Jan-2022].