



23456				Silesian University of Technology			
			course	group	section		
	SSI		BIAI	2	2		
	Nikodem Wspaniały	lecturer:		GB			
	Alan Pawleta						
<p align="center"><i>Final report</i></p>							
<p align="center">Brain tumor model recognizer</p>							
Date		04.07.2024					

CONTENT

1. Introduction	3
2. Data Collection and Preprocessing	3
3. Possible Approaches to Solve the Problem	5
4. Training models	5
5. CNN Model	6
5.1 Model Architecture	6
5.2. Results	7
6. VGG16-based Model	9
6.1. Model Architecture	9
6.2. Results	10
7. InceptionV3-based Model.....	11
7.1. Model Architecture	11
7.2. Results	13
8. Model Comparison.....	14
9. Testing	14
10. Conclusion	15
11. References.....	15

1. Introduction

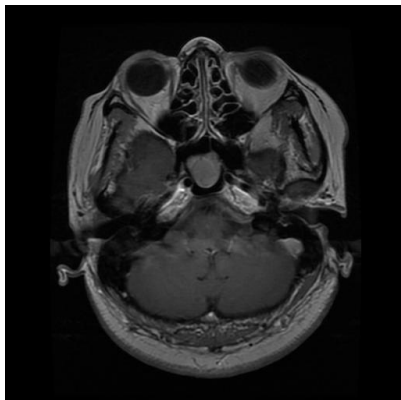
The aim of this project was to develop a Convolutional Neural Network (CNN) model for the classification of brain tumors. The model was designed to identify four categories of brain tumors: pituitary, no tumor, meningioma, and glioma. The project utilized Keras and TensorFlow, popular libraries in the field of deep learning, to implement and train the model.

2. Data Collection and Preprocessing

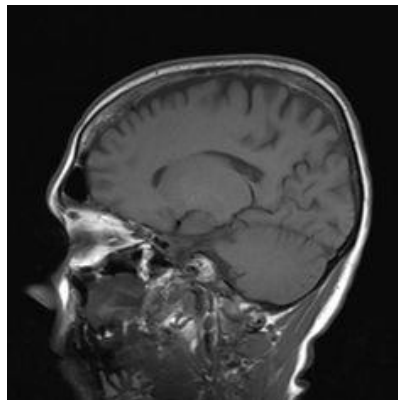
The dataset used for training and testing the model was sourced from Kaggle database. The dataset comprised MRI images categorized into four classes: pituitary, no tumor, meningioma, and glioma.

Sample pictures:

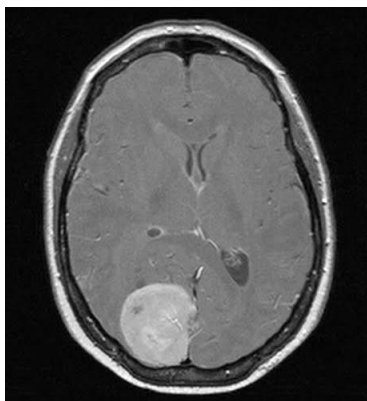
PITUITARY



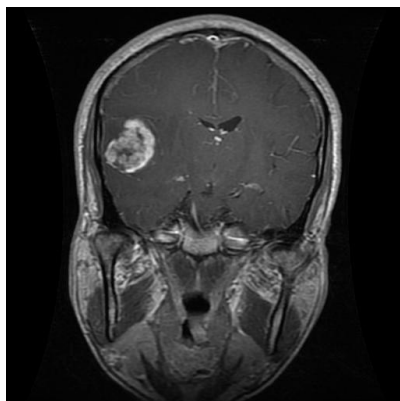
NO TUMOR



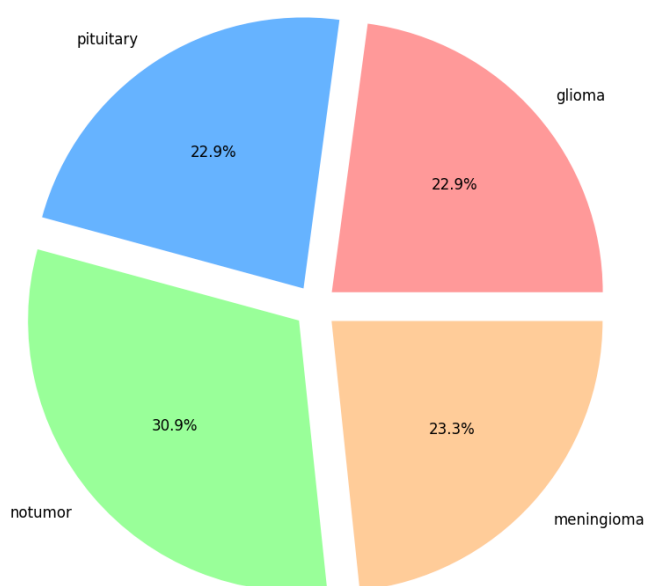
MENINGIOMA



GLIOMA



Dataset ratio:



Preprocessing steps included:

- **Resizing Images:** All images were resized to a uniform size to maintain consistency in input data dimensions.
- **Normalization:** Pixel values were normalized to the range [0, 1] to ensure faster convergence during training.
- **Data Augmentation:** Techniques such as rotation, flipping, and zooming were applied to augment the training data and prevent overfitting.

3. Possible Approaches to Solve the Problem

Several approaches can be taken to solve the problem of brain tumor classification:

1. **Traditional Machine Learning Approaches:**
 - **Pros:** Easier to interpret, requires less computational power.
 - **Cons:** Requires manual feature extraction, typically lower accuracy for complex image data.
2. **Pre-trained Deep Learning Models (Transfer Learning):**
 - **Pros:** Leverages existing models, saves time and computational resources, achieves higher accuracy.
 - **Cons:** Requires fine-tuning, larger model size, potential overfitting.
3. **Custom CNN Model:**
 - **Pros:** Tailored architecture for specific tasks, can achieve high accuracy, flexible to modify.
 - **Cons:** Requires substantial computational resources and time, needs a large dataset, complex to design.

Given the complexity of medical imaging and the availability of computational resources, we chose to develop a custom CNN model tailored to our specific task of brain tumor classification.

4. Training models

Models were compiled using the Adam optimizer with a categorical cross-entropy loss function. The training process involved the following steps for every model:

- **Splitting the Data:** The dataset was split into training, and test sets in a ratio of 81:19.

- **Training:** The model was trained for 50 epochs with a batch size of 32. Early stopping was implemented to prevent overfitting by monitoring the validation loss.
- **Evaluation:** The performance of the model was evaluated on the test set using metrics such as accuracy, precision, validation accuracy, validation precision.

5. CNN Model

5.1 Model Architecture

The CNN model was built using the Keras API with TensorFlow as the backend. The architecture of the model is as follows:

```
def build_model():
    model1 = Sequential()
    model1.add(Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3)))
    model1.add(MaxPooling2D((2, 2)))
    model1.add(Conv2D(64, (3, 3), activation='relu'))
    model1.add(MaxPooling2D((2, 2)))
    model1.add(Conv2D(32, (3, 3), activation='relu'))
    model1.add(Flatten())
    model1.add(Dense(16, activation='relu'))
    model1.add(Dense(4, activation='softmax'))
    model1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model1

model1 = build_model()
model1.summary()
```

Where IMAGE_SIZE is 128

The architecture of the model includes:

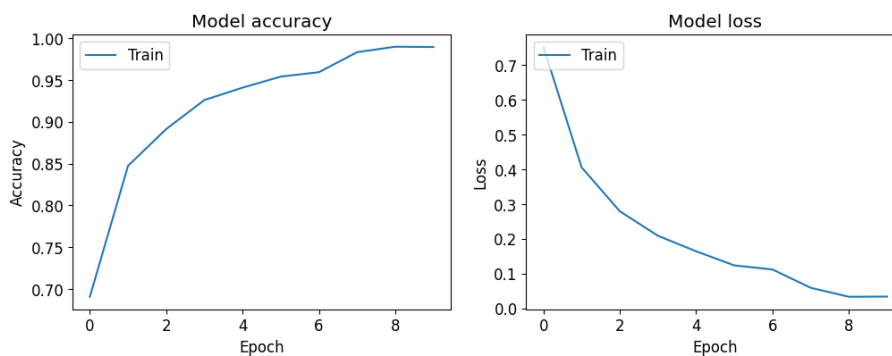
- **Input Layer:** The input layer expects images of size (IMAGE_SIZE, IMAGE_SIZE, 3).
- **Convolutional Layer 1:** The first convolutional layer has 32 filters of size 3x3 and uses the ReLU activation function. This layer extracts basic features from the input images.
- **Max Pooling Layer 1:** The first max pooling layer reduces the spatial dimensions of the feature maps by a factor of 2, helping to reduce computational complexity and extract dominant features.
- **Convolutional Layer 2:** The second convolutional layer has 64 filters of size 3x3 with ReLU activation. This layer captures more complex features from the images.

- **Max Pooling Layer 2:** The second max pooling layer further reduces the spatial dimensions.
- **Convolutional Layer 3:** The third convolutional layer has 32 filters of size 3x3 with ReLU activation. This layer refines the feature extraction process.
- **Flatten Layer:** This layer flattens the 3D feature maps into a 1D vector, preparing it for the fully connected layers.
- **Dense Layer 1:** A dense (fully connected) layer with 16 units and ReLU activation. This layer processes the extracted features for classification.
- **Output Layer:** The output layer has 4 units (corresponding to the four classes) with a softmax activation function, which provides the probability distribution over the classes.

The model is compiled using the Adam optimizer and the sparse categorical cross-entropy loss function, with accuracy as the evaluation metric.

5.2. Results

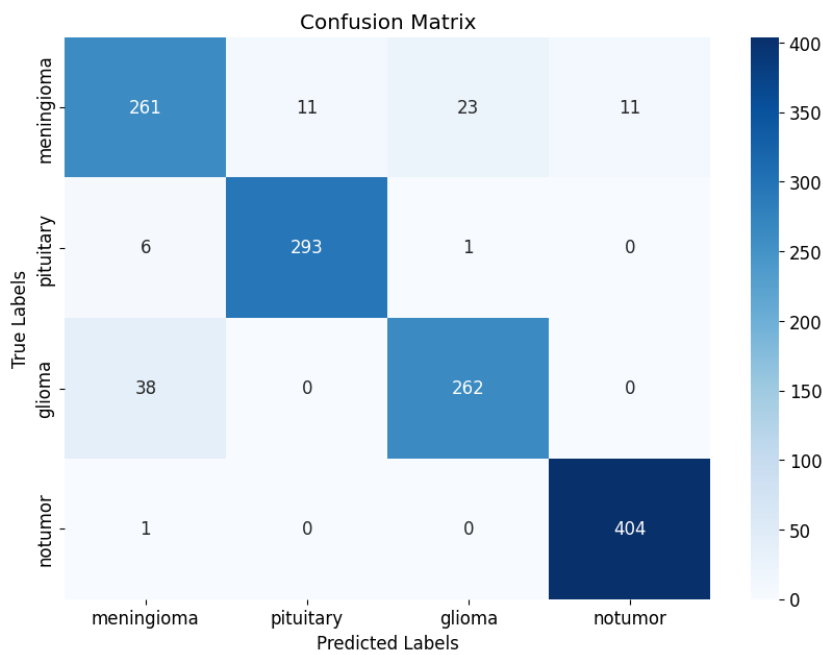
The trained model achieved an accuracy of 95% on the test set. The confusion matrix indicated that the model performed well across all categories with minor misclassifications.



The detailed performance metrics are as follows:

- **Accuracy:** 95%
- **Loss:**0.04%
- **Validation_accuracy:** 90%
- **Validation_loss:** 0.07%
- **Prediction:**
 - Meningioma: 0.92%
 - Pituitary: 0.96%
 - Glioma: 0.88%
 - No tumor: 0.99%

The presented confusion matrix is a valuable tool for analysing the classification results of the model. Each cell in the matrix indicates the number of instances where the model's predictions align or diverge from the actual class labels, providing a visual representation of the model's accuracy and highlighting patterns of misclassification across different classes.



6. VGG16-based Model

6.1. Model Architecture

The VGG16-based model was built using the Keras API with TensorFlow as the backend. The architecture of the model is as follows:

```
base_model2 = VGG16(input_shape=(IMAGE_SIZE,IMAGE_SIZE,3), include_top=False, weights='imagenet')

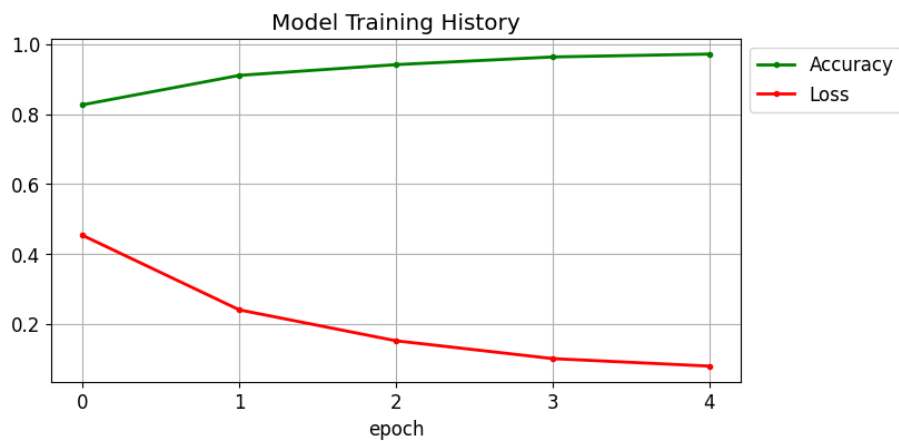
for layer in base_model2.layers:
    layer.trainable = False

base_model2.layers[-2].trainable = True
base_model2.layers[-3].trainable = True
base_model2.layers[-4].trainable = True

model2 = Sequential()
model2.add(Input(shape=(IMAGE_SIZE,IMAGE_SIZE,3)))
model2.add(base_model2)
model2.add(Flatten())
model2.add(Dropout(0.3))
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.2))
model2.add(Dense(len(unique_labels), activation='softmax'))
```

- **Base Model:** VGG16 pre-trained on ImageNet, excluding the top classification layer.
 - Layers -2, -3, and -4 are trainable, while the rest are frozen.
- **Input Layer:** The input layer expects images of size (128, 128, 3).
- **Flatten Layer:** Flattens the 3D feature maps into a 1D vector.
- **Dropout Layer 1:** A dropout layer with a rate of 0.3 to prevent overfitting.
- **Dense Layer 1:** A dense (fully connected) layer with 128 units and ReLU activation. This layer processes the extracted features for classification.
- **Dropout Layer 2:** A dropout layer with a rate of 0.2.
- **Output Layer:** The output layer has units corresponding to the number of classes (equal to the length of `unique_labels`) with a softmax activation function, providing the probability distribution over the classes.

6.2. Results



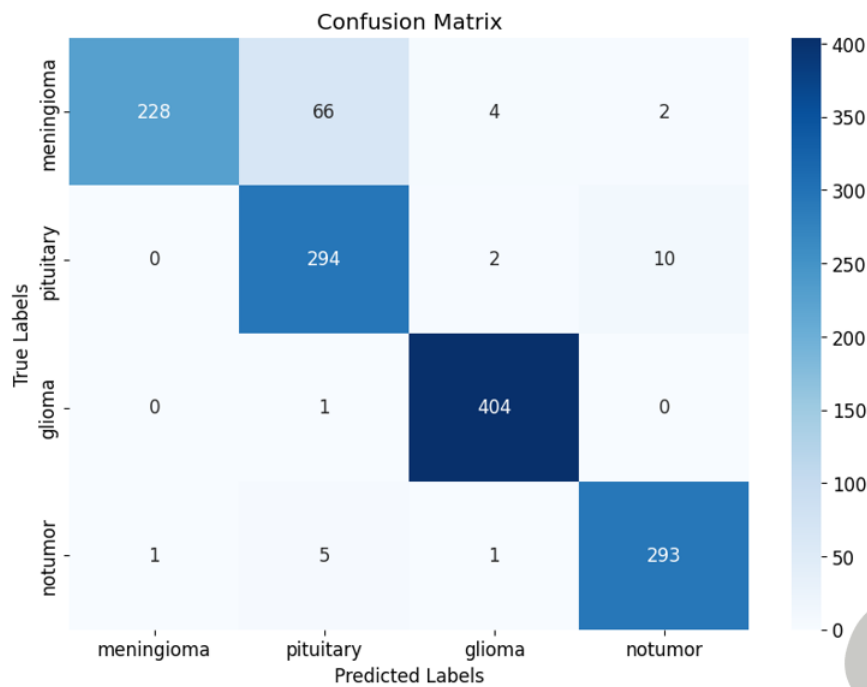
The trained model achieved the following performance metrics:

- **Accuracy:** 94%
- **Loss:** 0.01
- **Validation Accuracy:** 93%
- **Validation Loss:** 0.07
- **Prediction:**
 - **Menigioma:** 0.80
 - **Pituitary:** 0.96
 - **Glioma:** 1.00
 - **No tumor:** 0.98

The confusion matrix for the VGG16-based model indicates strong performance across all categories with minor misclassifications.

Z komentarzem [NW1]: *

Z komentarzem [NW2]: *



7. InceptionV3-based Model

7.1. Model Architecture

The InceptionV3-based model was built using the Keras API with TensorFlow as the backend. The architecture of the model is as follows:

```

base_model3 = InceptionV3(input_shape=(IMAGE_SIZE,IMAGE_SIZE,3), include_top=False, weights='imagenet')

for layer in base_model3.layers:
    layer.trainable = False

base_model3.layers[-2].trainable = True
base_model3.layers[-3].trainable = True
base_model3.layers[-4].trainable = True
base_model3.layers[-5].trainable = True

model3 = Sequential()
model3.add(Input(shape=(IMAGE_SIZE,IMAGE_SIZE,3)))
model3.add(base_model3)
model3.add(Flatten())
model3.add(Dropout(0.3))
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.2))
model3.add(Dense(256, activation='relu'))
model3.add(Dropout(0.2))
model3.add(Dense(len(unique_labels), activation='softmax'))

model3.summary()

```

- **Base Model:** InceptionV3 pre-trained on ImageNet, excluding the top classification layer.
 - Layers -2, -3, -4, and -5 are trainable, while the rest are frozen.
- **Input Layer:** The input layer expects images of size (128, 128, 3).
- **Flatten Layer:** Flattens the 3D feature maps into a 1D vector.
- **Dropout Layer 1:** A dropout layer with a rate of 0.3 to prevent overfitting.
- **Dense Layer 1:** A dense (fully connected) layer with 128 units and ReLU activation.
- **Dropout Layer 2:** A dropout layer with a rate of 0.2.
- **Dense Layer 2:** A dense layer with 256 units and ReLU activation.
- **Dropout Layer 3:** A dropout layer with a rate of 0.2.
- **Output Layer:** The output layer has units corresponding to the number of classes (equal to the length of `unique_labels`) with a softmax activation function, providing the probability distribution over the classes.

The model is compiled using the Adam optimizer and the categorical cross-entropy loss function, with accuracy as the evaluation metric.

7.2. Results



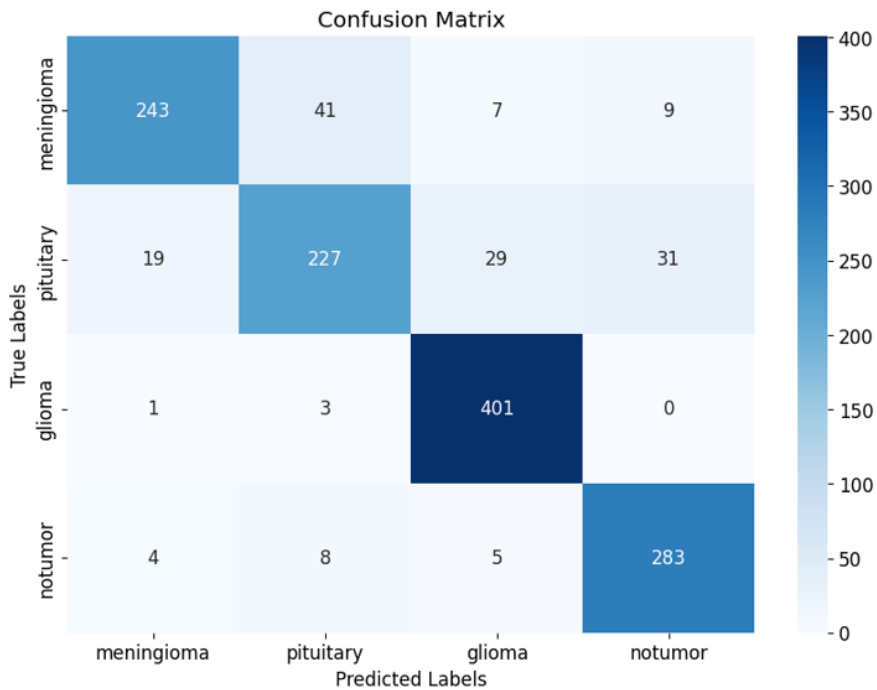
The trained model achieved the following performance metrics:

- **Accuracy:** 88%
- **Loss:** 0.02
- **Validation Accuracy:** 88%
- **Validation Loss:** 0.04
- **Prediction:**
 - **Menigioma:** 0.81
 - **Pituitary:** 0.88
 - **Glioma:** 0.91
 - **No tumor:** 0.91

The confusion matrix for the InceptionV3-based model shows excellent performance with very few misclassifications, highlighting the model's robustness.

Z komentarzem [NW3]: *

Z komentarzem [NW4]: *

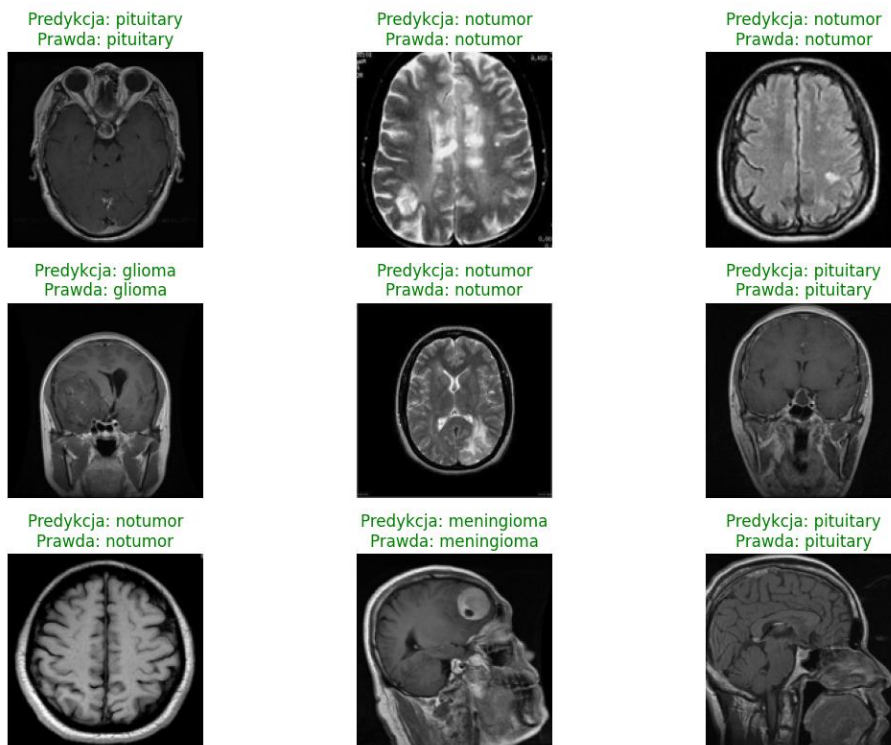


8. Model Comparison

9. Testing

To further evaluate the model's robustness and performance, we developed a simple application that accepts MRI images, loads the trained model, and classifies the images into one of the four categories. This application was used to test the model on new images, demonstrating its practical applicability and providing insights into its real-world performance. The results from these tests reaffirmed the model's high accuracy and robustness, though some areas for improvement were identified, particularly in distinguishing between similar tumor types.

Example of testing:



10. Conclusion

This project successfully developed a CNN model for the classification of brain tumors using Keras and TensorFlow. The model demonstrated high accuracy and robustness, making it a valuable tool for assisting medical professionals in diagnosing brain tumors. Further improvements and more extensive testing on larger datasets could enhance the model's performance and reliability.

11. References

- Keras Documentation: <https://keras.io/>
- TensorFlow Documentation: <https://www.tensorflow.org/>

- Dataset Source: [kaggle](#)
- Github repository:
<https://github.com/NikodemWspanialy/BrainTumorModelTraining>