UNIVERSITY OF OSLO

Project 3

FYS-STK4155 – Applied Data Analysis and Machine Learning

Hishem Kløvnes

Gerlind Deschner

Juan Manuel Scarpetta

December 13, 2024

Abstract

Machine learning has emerged as a helpful tool in medicine. Particularly in automating complex tasks, like for instance, classification tasks. In this project, we developed and evaluated machine learning models for classifying brain MRI images into four categories: glioma, meningioma, pituitary tumors and no tumors. We delved into the cores of convolutional neural networks (CNNs), where we attempted classification with pre-trained models, custom trained models and custom-made models. By comparing the performance of traditional methods like Random Forests and Gradient boosting with convolutional neural networks (CNNs), we obtained significant insight into the different methods of the use of CNNs. We understood both strength and weaknesses of the different models we used. Some models, like MobileNet, can be extremely efficient, while still achieving accuracy scores over 90%, when coupled with gradient boosting. Other more time consuming models, like the custom made ones, or the Densenet model achieved remarkable scores, with over 97% in accuracy, without the need for gradient boosting.
This study underlines the potential of machine learning in medical diagnostics, demonstrating how tailored CNN architectures and robust hyperaparameter tuning, can provide highly accurate and efficient models. With further advancements, these techniques could significantly improve diagnostic and reduce workloads for medical professionals.

Contents

1 Introduction

In recent years, machine learning has proven to be a powerful tool in various medical fields, showing great promise in assisting healthcare professionals with diagnosis and releaving some of the workload of medical workers. One area where these advancements have demonstrated significant potential is in the classification of tumors using magnetic resonance imaging (MRI). Early and accurate diagnosis is important for planning treatment and improving patient outcomes. Machine learning models, particularly convolutional neural networks (CNNs), have shown remarkable success in image classification tasks, making them well-suited for automating and enhancing the diagnostic process.

The primary goal of this project is to develop a CNN model capable of classifying MRI brain images into four categories: glioma, meningioma, pituitary tumors, and no tumor. By using advanced machine learning techniques, including CNNs, random forests, and boosting methods, we aim to evaluate the effectiveness of these models in the context of medical imaging. Each of these methods has its own strengths, and through comparison, we will identify the most suitable model for this specific task. In addition to classification accuracy, we will assess the models using multiple evaluation metrics, such as F1 scores, R^2, and mean squared error (MSE), to provide a comprehensive understanding of their performance. Tracking the training history will offer insights into the optimization process, helping us to refine and select the best-performing model.

To find the optimal model with the highest accuracy in the MRI classification task, we explored numeral networks from the `Pytorch` [4] package. The aim was to identify a model that balances performance and computational efficiency for this application.

This project aims not only to contribute to the growing field of machine learning in medical applications but also to demonstrate the effectiveness of machine learning models in making reliable predictions.

2 Data

For the development of this project, the Kaggle Brain Tumor MRI by Masoud Nickparvar [14] dataset is taken as the main data. This dataset is a combination and collection of three pre-existing datasets containing different brain images where different kinds of tumors are evidenced through MRI. This dataset has a total of 7023 images, divided into 5712 training images and 1311 test images. The natural objective of this Kaggle dataset is to develop and compare different classification models that are able to efficiently distinguish the 4 classes of tumors: glioma - meningioma - non-tumor and pituitary. It should be noted that each of these images are of different sizes, so when implementing the classification models it is necessary to perform data preprocessing to rescale the samples to a standard size. Some image samples from this dataset are illustrated in Figure 1.

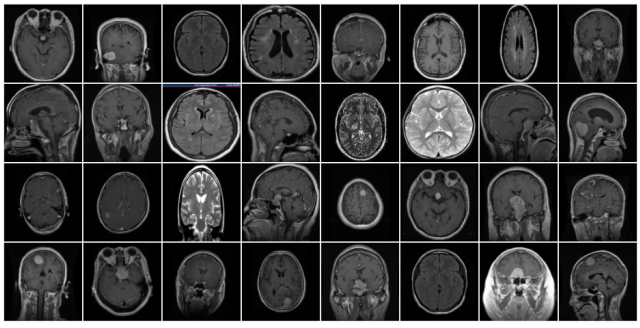


Figure 1: Sample train batch of images from the Kaggle Brain Tumor MRI dataset.

3 Methods

This section covers the utilized methods, their theoretical aspects, particular models and other results. Relevant metrics and functions are introduced.

3.1 Convolutional Neural Networks

In this section we introduce Convolutional Neural Networks (CNNs) which are a popular image classification method in deep learning. The structure, strengths and weaknesses are shortly presented. They can be discussed in far greater detail, for example see the book by Raschka et al. [16, Chapter 14], as they refer to many further literature.

CNNs are a special case of a feedforward neural network, which we discussed in the report for project 2 [13] and also shortly recap in Section 3.7.3. In particular, these networks utilize that neighboured pixels have meaningful relationship with another. There are characteristic layers a CNN is build of: convolutional layers, pooling layers (also called subsampling layers), and fully connected layers.

The convolutional layers perform the convolution of a filter (also called kernel) with the subimages of the same size. For instance, when using an 4×4 image X and a 3×3 filter F, we could perform the convolution with a subimage four times by starting at the upper left corner and moving it to the right for all the rows as often as possible. As a result, we get a 2×2 image. This result is defined as

$$Y = X \star F, \quad Y_{i,j} = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} X_{i-k,j-l} F_{k,l}$$

where the defined defined ranges of k, l, i and j are in practice only considered for feasible indices.
In order to achieve an image of the same size after the convolution step, padding can be used. A popular method is zero-padding. For instance, if we add zero pixels around our 4×4 image, we get a 6×6 image and after applying the 3×3 filter we get a desired output of size 4×4. The size of the filter is often chosen to be very small, for instance 1×1 or 3×3, especially compared to the initial image size, as this is often much larger, for instance 128×128. Moreover, above we explained that we can move this filter by one pixel and apply it each time. However, we could also use a so called stride to move the filter more than one pixel horizontally and vertically. The weights for the convolution and the paramters are optimized in the training.

Often a pooling layer follows a convolutional layer. These, however in the contrast to the convolutional and fully connected layers, have no parameters that we optimize. They are employed to reduce the feature size of the features extracted from the convolutional layer. The two most popular types are max-pooling and mean-pooling. These depend on a pooling size which specifies how large the subimage is for which we apply the pooling function. For instance, if we consider a 4×4 image and apply a 2×2 max-pooling, we get as a result a 2×2 image with for which the pixels corresponds to disjoint 2×2 subimages and the value in that pixel is the maximum value that occurs in the corresponding subimage. For mean pooling, this value is defined as the mean value of the values in the corresponding subimage. Max-pooling is robust in the sense that small changes in the image may not even affect its outcome. Moreover, pooling can also be conducted with overlapping, so that the pooling subimages are not disjoint.

After some convolution and pooling layers a fully connected layer follows to conclude to the final classification.
Training and prediction is conducted as we described in the report for project 2 [13] using gradient methods.
These and further details can be found in the book by Raschka et al. [16, Chapter 14].

3.2 Examples of Networks

Now as we are familiar with the general structure of a CNN, some particular models are introduced, which turned out to be very useful.

The MobileNet is a small network that was developed for the efficient computation on mobile or embedded devices. It uses depthwise convolution which is different from the standard convolutional layer. The subsequent versions 2 and 3 utilize further structures to improve the performance [2, Section 3.5.2.].
Similarly, EfficientNetV1 is also a smaller and efficient model, which however requires careful training to reach high accuracy. It is motivated by the relationship between the depth and width of a network [2, Section 3.5.7.].
In comparison, the ResNet model is a deeper network which belongs to the class of residual learning networks. It easily gains accuracy by its added layers but the deepness of the network is a great risk for overfitting [2, Section 3.3.1.].
The DenseNet model faces the problematic of vanishing gradients and brings a remarkable improvement by connecting not only the convolutional layers to the next, but by connecting every layer to every other layer in the network [2, Section 3.3.4].

The PyTorch library for python, for instance, also offers pretrained paramteres for the mentioned models [4]. Those can in particular be useful to extract some valuable features from the image data and then apply other methods. Those that are not suitable for image data, but for tabular data. Such a method is for instance random forests.

3.3 Random Forests

The random forests method is an ensemble method motivated by Bagging (bootstrap aggregation). Bagging

aggregates the predictors that are each trained on a bootstrap sample. These predictors, or weak learners, are often chosen to be trees, as these are methods with high variability, which the bagging is able to reduce. However, it has the weakness that if the bootstrap samples are correlated, the have a negative impact on the variance of the variance of the bagging. The random forests construct the trees so that they are less correlated. This is achieved by taking a random selection of variables for each tree splitting step and choosing the best of these variables to split on. The trees are constructed with respect to a particular depth or minimum node size. The minimum node size specifies how many observations have to be associated to every node for the tree construction to stop. If this size is small, a large and complex tree is constructed and a large size creates simpler trees. Often a larger size works good, but if a smaller size works better, this can also imply that the underlying structure is more complex.
These and further details can be found in the book by Hastie et. al [7, Chapter 15].

3.4 Gradient Boosting

Boosting is a powerful method which also has been argued to be the best classifier before neural networks became more popular. The central idea in boosting methods is to perform gradient descent, but instead of using the gradient, a weak learner is trained to approximate the gradient. Additionally, when adding the predictor, only a certain fraction of it can be used. This so called boosting step size chosen to be between zero and one and it enables slow learning. It has the main advantage that it avoids overfitting. In classification popular loss functions are the negative log-likelihood or the exponential loss.
For these and more details see the book by Hastie et al. [7, Chapter 10].
An outstanding performance can be achieved using the very efficient and scalable gradient boosting algorithms provided by the Extreme Gradient Boosting (XGBoost) library [6, 3]. This method comes with a variety of hyperparameters also including L1-norm and L2-norm regularization.

3.5 Dimensionality Reduction

In highdimensional settings it is practical to reduce the numbers of parameters to be able to use certain methods or reduce the complexity. We already saw that the Lasso Regression performs such model selection. Here we introduce the Principal Component Analysis (PCA).

Consider a design matrix $X \in \mathbb{R}^{n,p}$, using the singular value decomposition, it can be written as $X = U\Sigma V^T$ for $U \in \mathbb{R}^{n,n}, V \in \mathbb{R}^{p,p}$ orthogonal matrices and $\Sigma \in \mathbb{R}^{n,p}$ diagonal matrix with entries $\sigma_1, \dots, \sigma_d, 0, \dots, 0$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0$, the principal components, for d the rank of the matrix X. As we can write $X^\top X = V\Sigma^2 V^\top$ which is the eigen-decomposition of $X^\top X$, we have $(X^\top X)V = V\Sigma^2$, which gives $(X^\top X)v_i = \sigma_i^2 v_i$, where v_i is the i-th column of the matrix V and it is called the i-th principal component direction. Correspondingly $Xv_i = z_i$ is called the i-th principal component. By considering the discrete sample variance, we get that the first principal component z_1 corresponds to the direction which maximizes the variability in the data. Correspondingly, the second principal component corresponds to the direction which is orthogonal to the first principal direction that maximizes the variability in the data. In the same matter, the other principal components maximize orthogonally the variability.
To perform model selection, which is essentially also feature extraction in this case, the first $M \leq p$ principal components can be chosen as they explain most of the variability in the data.
The scikit-learn python library offers an implementation of this method [5].
These and further details can be found in the book by Hastie et al. [7, Section 3.4, 3.5].

3.6 Model Performance Measures

This section introduces the main metrics used to evaluate performance. First considering quantitative and then qualitative data.

3.6.1 Mean Squared Error and Coefficient of Determination

For qualitative data the mean squared error (MSE) and coefficient of determination (R2-score) are considered.
As already introduced in the first project, the mean squared error (MSE) is the average squared error of the residuals

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2,$$

and thus, it is non-negative but uses a different scaling than the target variable and is greatly impacted by outliers. Since it is an average, it can be used to compare the results between sets of different sizes.
The coefficient of determination is an advantageous transformation of the mean squared error. It is upper bounded by one and in case it is equal to one, we have that the response variable is perfectly explained by the model utilized for the prediction. The higher the score, the better. However, if the score takes negative values, this generally represents a poor performance of the model as it implies that the simple average of the true response variables yields a better mean squared error than the suggested prediction.
These details can be found in the first project report [12].

3.6.2 Accuracy and $F1$-Score

As we consider a classification task for this project, in particular metrics for qualitative data are of importance. We shortly introduce the F1-score and the accuracy.

Consider the classification setting of four classes where the true k-th response variable is one if it belongs to class k, and zero if it does not belong to class k for

$k \in \{1, 2, 3, 4\}$ represents the four classes. The prediction of an observation is then a 4-dimensional vector which is transformed to have one as entry for its largest entry. The other entries are set to zero. This vector for the i-th observation is now denoted by $\hat{y}_i$, and $c(\hat{y}_i)$ refers to the predicted class. After transforming the prediction, we can compute the accuracy. This is defined as

$$\text{accuracy}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}(c(\hat{y}_i) = y_i)$$

for the true response values y which has $y_i = c$ if c is the class for the i-th observation. The function $\mathbb{1}$ here is just the characteristic function. So, if the accuracy is equal to one, then we managed to predict the outcome perfectly. Apart from that it takes values in $[0, 1]$, where a score of zero means that nothing was predicted correct. However, one has to keep in mind that this metric only yields a general evaluation, as it does not consider the true not transformed predictions. Those have to be analysed separately to get a better understanding of the behaviour of the selected classification method.

To define the $F1$-score we first introduce the precision (PRE) and recall (REC). The recall is defined as the true positive rate which is the number of true positive (TP) divided by the number of actually positives, which is thus the some of true positives and false negatives (FN), which means that it is defined as

$$REC = \frac{TP}{TP + FN}.$$

The precision defines the relationship of the predicted positive and true positives

$$PRE = \frac{TP}{TP + FP},$$

where FP denoted the false positives. Together, they define the $F1$-score

$$F1 = 2\frac{PRE \cdot REC}{PRE + REC}$$

which gives the advantage of considering the false negatives and false positives in one metric compared to only considering hte precision or recall.

These and further details can be found in Chapter 3 and 6 of the book by Raschka et al. [16, Chapter 3, 6] and in the report for the second project [13].

3.7 Other Methods

This section gives a short recap other utilized methods, which were already introduced in the previous project reports [12, 13].

3.7.1 Resampling Methods

The k-fold crossvalidation is a resampling method used to choose hyperparameters over a given grid of possible choices by determining the average of a metric for multiple runs. By randomly splitting the data in k sets and then considering each of the sets once as a test set and the rest as the training set. For each of those k splits, we can then train a model and evaluate on the test sets, of which we take the average in the end. Usually the k is chosen to be five or ten [7, Section 7.10]. It is a computationally expensive method, easy to implement and can be applied to a variety of models. In contrast to the for instance bootstrap methods, all the offered data is taken into account.
These details can also be found in the report for the second project [13].

3.7.2 Logistic Regression

In this section we shortly recap the classification method logistic regression.
In the following, to introduce the idea we describe the case of two classes $\{0, 1\}$. However, this can simply be generalized to more than 2 classes. The logistic regression works with the probabilities p_i that represent the probability $\mathbb{P}(Y_i = 1 \mid X = x_i)$, where Y_i is the random variable of the outcome and X of the features. For example, if the data x_i belongs to the class 1, the $p_i = 1$, else it is 0. Using the logit function $\text{logit}(p) = \log(\frac{p}{1-p})$ these probabilities can be transformed to be on the real line $\mathbb{R}$. However, this is not possible as they cannot take infinte values. So, the projections of the "infinite" points onto the logit function are used. This logit function over the feature axis is linear, so we can describe it as

$$\text{logit}(p) = \beta_0 + \beta_1 x$$

where we assume for sake of simplicity that we only consider one feature. the generalization to multiple features is trivial. Using the inverse, the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$ (which is later also referred to as sigmoid), we achieve that

$$\mathbb{P}(Y = y \mid x) = \sigma(\beta_0 + \beta_1 x)^y (1 - \sigma(\beta_0 + \beta_1 x)^{1-y}$$

for a feature x and a response $y \in \{0, 1\}$. We want to maximize these probabilities for the observations by considering different β_0, β_1. Maximizing the likelihood, by applying the logarithm, is equivalent to

$$\hat{\beta}^{\log} = \arg\max_{\beta} \sum_{i=1}^{n} \big(y_i(\beta_0 + \beta_1 x_i)) - \ln(1 + e^{\beta_0 + \beta_1 x_i})\big).$$

To find the optimum coefficients, gradient methods can be utilized. When predicting for two classes $\{0, 1\}$ for example, the prediction is class 1 if the value is closer to one than to zero, and else it is class 0. As the model still depends on a linear relationship, it offers interpretablility. To avoid overfitting, also L1- and L2-norm penalties can be added.
These and further details can be found in the book of Hastie et al. [7, Section 4.4] or Rascka et al. [16, Chapter 3]. These details can also be found in our second report [13].

3.7.3 Feedforward Neural Network

This section shortly repeats some detailsof the feedforward neural network.

These networks consist of an input layer, hidden layers and an output layer and are also called multilayer perceptrons (MLPs). Each layer consists of a certain number of nodes and the input layer has a node for every feature in our data and it forwards these features to the subsequent first hidden layer. In this layer, every node then takes an affine combination of the data from the previous layer using weights and biases and transforms the result using an activation function. This activation function is specific for the layer, but the weights and biases are specific for each node in that layer. These results are then the input for the next layer. The number of nodes in the output layer depends on the response variable. In case of classification and a four qualitative variables, we have a four nodes in the output layer. This now explained the feedforward step of a neural network and the general setting we work with.
The motivation behind this structure is the universal approximation theorem, which states that a feedforward neural network with just a single hidden layer containing a finite number of neurons can approximate a continuous multidimensional function to arbitrary accuracy, assuming the activation function for the hidden layer is a non-constant, bounded and monotonically-increasing continuous function" [8, Section 13.5].
However, the mentioned weights and biases have to be determined and this is accomplished using the backpropagation algorithm for a gradient method to optimize their values. The loss function with respect to which we want to optimize the weights and biases has to chosen. Loss functions for classification is mentioned in Section 3.7.5.
By optimizing the coefficients by using the gradients there can occur certain poor behaviors due to the structure of the network. The vanishing gradients problem is that the more layers we add to the network, the smaller the gradients get. In particular this can make a proper convergence impossible. In contrast, the exploding gradients problem, for which the gradients are very large can yield problems. Altogether, it must be considered that the different layers of a network converge differently.
The universal approximation theorem suggests that we should use some "non-constant, bounded and monotonically-increasing continuous" activation functions. Popular choices are the sigmoid (also called logistic function), the ReLu, leaky ReLu and the softmax function. These activation functions have their strenghts and weaknesses and have to be used cautiously as, for instance, the relu maps onto the non-negative numbers, the leaky relu onto the real numbers and the softmax and sigmoid only map onto the interval $[0, 1]$.

The main advantage of this method is that it allows for very complex and non-linear models that in practice can perform very well. However, its complexity is also its weakness as it is poorly interpretable and the tuning of the variety of hyperparameters is difficult.
These details can be found in more detail in the report for our second project [13]. Further details can also be found in the lecture notes by Hjorth-Jensen [8, 9] and in the book by Rascka et al. [16, Chapter 12].

3.7.4 Gradient Methods

Gradient methods form the heart of training neural networks. The gradient method utilized in this project is Adam, which we now very shortly recap. For more information about gradient methods and other algorithms see the report for project 2 [13].

In general, gradient methods optimize some particular weights with respect to a certain loss function by shifting the weights towards their negative gradient (which goes in the direction of the strongest decrease with respect to the loss function) with a step size which is called the learning rate.
When applying this to many observations it is very costly to compute the gradient. This motivates the "mini-batch" gradient descent (SGD), which for a specified number of iterations (also called epochs) computes the gradients batches of the data and updates the weights based on these gradients. The method is computationally much cheaper than the simple gradient descent.
The Adam method implements this, and moreover, it defines the learning rate so that it changes adaptively without introducing high computational costs. An adaptively changing learning rate enables better performance of the convergence. Adam, as the root mean squared propagation (RMS prop), uses the first and second momentum. The first momentum is a value in $[0, 1]$ and keeps track of the changes of the weights. The second momentum is the expectation of the squared gradient and it is used to scale the gradients. This especially leads to the learning rate being lower where the gradient length keeps staying large, and it increases in non-steep regions. Overall, it enables faster convergence. This however still imposes a bias, which is aimed to be corrected by the Adam method. Adam, in addition to the previously described method introduces an estimation of the moments. This results in a well balanced optimization algorithm which in practice showed an outstanding performance.

Details for this method can be found in the lecture notes and in the paper "Adam: A Method for Stochastic Optimization" [11] and in the lecture notes by Hjorth-Jensen [10].

3.7.5 Loss Functions

When training a method often a loss function has to be chosen. The cross-entropy, the negative log-likelihood and the softmax cost functions for classification are

now introduced.

The softmax function considers an m-dimensional vector z and is defined as

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}}$$

for $i \in \{1, \ldots, m\}$. This represents distributions, as the sum over all softmax transformations for an observation is equal to one and they are non-negative. It is also considered as activation function in neural networks.

For four classes and predictions $\hat{y}_i$ the cross-entropy loss is defined as

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i),$$

where y_i denotes the true observations. As we can notice, minimizing the cross entropy is then equivalent to maximizing the likelihood (which is the same as minimizing the negative log likelihood) and the predictions $\hat{y}_i$ correspond to the probabilities that an observation belongs to a certain class. So, in this setting values between zero and one should be chosen.
Moreover, we can add L1- and L2-norm regularization penalties to the loss function to introduce a bias while decreasing the variability among the solutions. This can help reducing overfitting.

These and further details can be found in the lecture notes by Hjorth-Jensen [8, 9] and in the book by Raschka et al. [16, Chapter 12].

4 Implementation

This section presents the implemented methods used for the classification problem. First we elaborate on the preprocessing of the data. Then the features extraction, classification and hyperparameter tuning methods are introduced. Finally we illustrate cases of custom trained weights.

4.1 Image Preprocessing

The images in the dataset have variying sizes, so the first step was to preprocess them. Each image was resized to be square by adding black padding until it formed a square shape. Afterwards, all images were scaled down to a consistent size of (128×128) pixels. Additionally, we applied data augmentation techniques such as random rotations, flips, and brightness adjustments. This was to increase the robustness of the model and prevent overfitting.

4.2 Feature Extraction

To extract meaningful features from the images, we employed pretrained CNNs, as these models are pretrained on large datasets, such as ImageNet, and have demonstrated great performance in image classifcation tasks. Using the pretrained models allowed us, in a time efficient manner, to capture patterns in the MRI dataset. By using the features extracted from these models rather than raw pixel values, we preserved the spatial relationships in the images, which are essential for accurate classification.

4.3 Classification Methods

Once the features were extracted from the images, we applied two classical machine learning algorithms, Random Forests and XGBoost. These models were selected because they are strong performers for structured data and can be efficiently trained on the features extracted by the CNNs.

- Random Forests: We used the RandomForestClassifier from the scikit-learn library. Random Forests are an ensemble method that builds multiple decision trees and combines their outputs to make more robust predictions. To tune the model, we experimented with hyperparameters, with the aim of finding a simple model, reducing risk of overfitting. However, this can lead to underfitting
- XGBoost: We also used XGBoost, a gradient boosting algorithm that builds decision trees to correct the errors of the previous trees. This method is widely known as an easy to use, and accurate method. The objective function we used in XGBoost was softmax, which is a suitcable method in multi-classifcation tasks. We tuned several hyperparameters for XGBoost, including number of trees, L1 and L2 regulizers and the minimum child weight. Again we had to keep in mind the relation between over and underfitting.

4.4 Hyperparameter Tuning

For both Random Forests and XGBoost, hyperaprameter tuning was performed using 5-fold-crossvalidation. This method splits data into five subsets, training the model on four of them and validating on the remaining one.

4.5 Custom Trained Weights

After evaluating how the classification methods performed with pretrained weights, we wanted to investigate how incorporating custom-trained weights might influence the models. We began by setting baseline performance metrics by training several different models on a training set and validating them on a test set. This allowed us to obtain model weights specifically tailored to our dataset. Then, we explored whether training the classifiers from scratch could further improve the accuracy of our models.

After applying both the RandomForestClassifier and XGBoost using pretrained CNN weights, we

compared their performance to determine the more effective method. Based on this comparison, we trained the top-performing CNN on our dataset and applied the best classifier to see if further training could improve classification accuracy.

The goal of this approach was two things: to find out whether training the CNN on our dataset could improve classification performance, and to identify which classifier, Random Forests or XGBoost, performed better in this context. This iterative process allowed us to systematically evaluate the impact of different components, such as pretrained vs. custom-trained CNNs and Random Forests vs. XGBoost, on overall performance.

4.6 Own CNN training and PCA

As a last implementation, to contrast with the results offered by the pre-trained networks, it was decided to implement a convolutional network architecture from scratch that allows to use the convolution operations on the images and connect them with a FFNN to perform the classification. Therefore, an SGD-Adam optimizer and a Cross Entropy Loss as an error function for training are used.

Another similar approach is employed using PCA, which allows to reduce the high dimensionality of the pixel data of the images to a reduced number of components represented in vector form in which the most important features of each sample are retained. After this, a much more manageable data structure is available to implement widely used classification models implemented in previous projects [12, 13] such as Logistic Regression, Random Forests and FFNNs.

5 Results

The results of the introduced implemented methods are now presented in this section. First we show the results for the classifiers on the pretrained weights, then for the custom trained convolutional neural networks, the costum trained networks with classifiers and the own convolutional neural network, and lastly for the principal component analysis.

5.1 Pretrained CNNs With Classifiers

In order to run the random forests method we first extracted features of the images using the MobileNetV2 model, which is a small CNN model. The random forests method was considered for 16, 32, 64, 128 and 256 weak learners and the trees had the default minimum node size of 2, meaning that they were complex trees. As a result, the testing accuracy was 82.4%. Considering Figure 2, we can also see that the random forests method overfits the data, but that the accuracy increases with the number of weak learners.

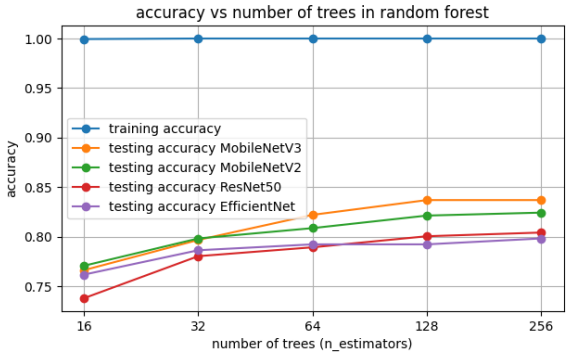


Figure 2: Accuracy of the random forests method for different numbers of learners using the features extracted from the pretrained MobileNetV2, ResNet50, EfficientNet-B0 and MobileNetV3 model.

Then we tried again a smaller network, the pretrained EfficientNet-B0, which again is a smaller network which can attain high accuracy. This also did not perform better than the MobileNetV2 model, as we can see in Figure 2.
Finally, when using the large pretrained MobileNetV3 model, we achieved an accuracy of almost 84%. Apart from that, this method performed similar to the others as seen in Figure 2. So we chose this model for random forests and boosting. Using crossvalidation (CV), the minimum node size of the trees for random forests is studied. As a result, clearly the most complex trees performed best as the minimum node size of two led to the highest accuracy as can be seen in Figure 3. This

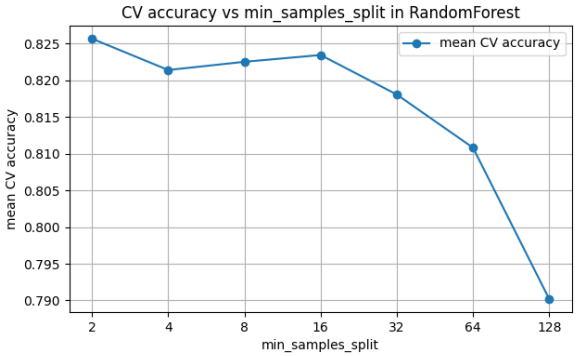


Figure 3: Accuracy of the random forests method using 5-fold crossvalidation for different numbers of minimum node sizes (min_samples_split) for the construction of the trees using the features extracted from the pretrained MobileNetV3 model.

makes sense, as our classification task is of a complex nature. Finally, we conduct crossvalidation to study the numbers of weak learners in the range from 50 to 400. The accuracy first increases with the number of estimators and then stabilizes from 300 trees on, so we chose 300 as the final number for our method. As a result, we get an accuracy of 84,31%. This result is also stated in Table 1. This result is still not good enough, but this method works extremely fast.
We already saw for random forests that the MobileNetV3

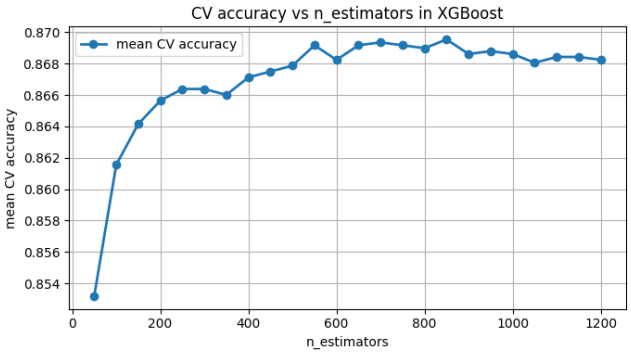


Figure 4: Accuracy of the XGBoost method using 5-fold crossvalidation for different numbers of weak learners using the features extracted from the pretrained MobileNetV3 model.

model for the feature extraction worked well, so we use this for boosting as well. By using then we applied crossvalidation to the XGBoost classifier to study different numbers of estimators. The results can be seen in Figure 4. The performance is great for 700 estimators as it stabilizes a bit there and gets a bit worse after 800 learners. So we fix the number of estimators to be 700 for now. By using this, we achieve a testing accuracy of 88.55% which is better than what we had with the random forests method. The training accuracy is 100% so we might have overfitting.

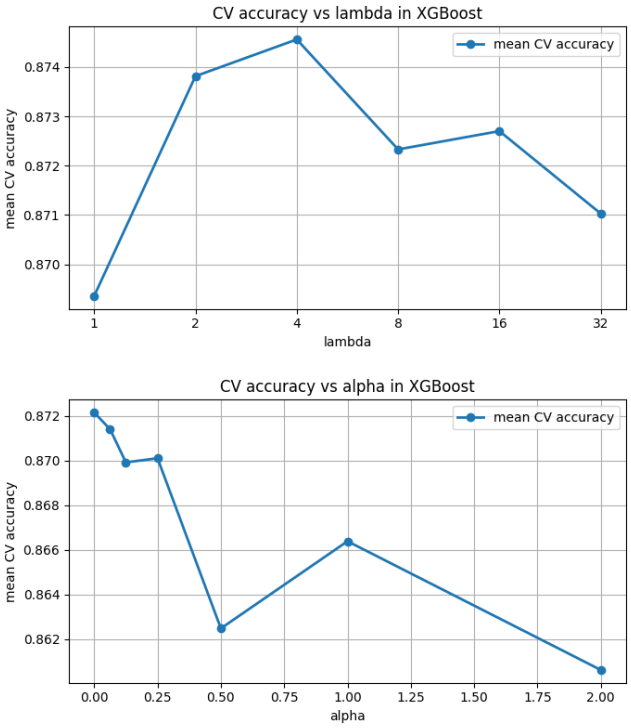


Figure 5: *Top:* Accuracy of the XGBoost method using 5-fold CV for the regularization parameter lambda (reg_lambda) in XGBoost using the features extracted from the pretrained MobileNetV3 model. *Bottom:* The same study but of the regularization parameter alpha (reg_alpha) with lambda fixed to 4.

Thus, the paramters alpha (reg_alpha) and lambda (reg_lambda) for the L1-norm and L2-norm regularization are considered using crossvalidation. First using again 5-fold crossvalidation, we consider multiple values for the regularization parameter for the L2-regularization. The default value is one, so we try larger values as these correspond to more regularization. The results are shown in the top in Figure 5.
The best performance is achieved for the parameter equal to four. Thus, we fix this parameter and study next the L1-regularization parameter alpha (reg_alpha). The default value for this parameter is zero, so we increase it from zero to see how the L1-regularization behaves using 5-fold crossvalidation. In the bottom in Figure 5 we can see that no L1-regularization seems to perform best. So we choose the parameter to be zero.
Next, we study the minimum child weight which is similar to the minimum node size in the random forests method. The default value is again one, so we study some larger values. The highest accuracy is attained for the weight of four so this parameter is now also fixed.
Using all the fixed parameters for the XGBoost classifier we get a training accuracy of 100% and a testing accuracy of 89,37% which is indeed better than the accuracy we achieved using random forests. Those results are summarized in Table 1, the training accuracies for both are still however 100%, which indicates that both methods still overfit.
Finally, again a crossvalidation over different numbers of estimators was conducted. The results are shown in Figure 6. Strangely the highest accuracy is attained for twenty estimators. Training such an XGBoost model now with all the previous studied parameters and twenty estimators gives a testing accuracy of only 84,68% and a training accuracy of 99,78%.

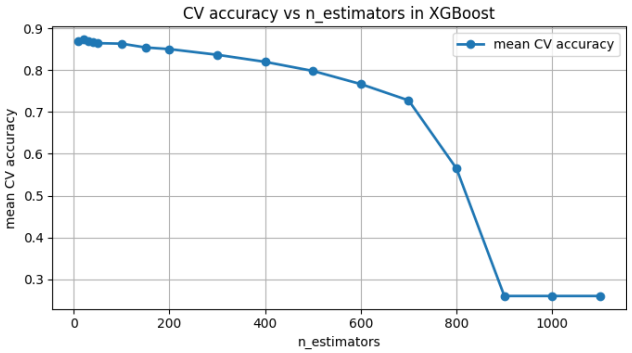


Figure 6: Accuracy of the XGBoost method using 5-fold CV for estimators in XGBoost using the features extracted from the pretrained MobileNetV3 model.

	random forests	gradient boosting
testing accuracy	84,31 %	89,37%

Table 1: Testing accuracies for the random forests and gradient boosting method for features extracted by the pretrained MobileNetV3 model.

5.2 Custom Trained CNNs Without Classifiers

Proceeding with training our own models, instead of using pretrained weights, we used the following parameters:

- Learning rate: 0.001
- Batch size: 32
- Optimizer: Adam
- Image Size: (128 × 128)
- Epochs: 20

Due to the time it takes to train CNN networks on such a large dataset, we had to narrow down which models we wanted to train, with only one optimizer. We narrowed it down to DenseNet, ResNet, EfficientNet and MobileNetV3.

Plain classes	NT	GT	MT	PT
Densenet	0.98	0.96	0.95	0.98
Efficientnet	0.98	0.94	0.90	0.95
Resnet	0.94	0.92	0.85	0.96
MobilenetV3	0.96	0.88	0.84	0.94

Table 2: Model performance on various tumor types (NT: No Tumor, GT: Glioma Tumor, MT: Meningioma Tumor, PT: Pituitary Tumor). Scores determined by f1 scores, from classification reports of the models performances. Full classification reports, can be found in appendix: 5, 6, 7, 8.

Looking at the scores from 2, we saw overall very good scores from the models performing on the test dataset. The absolute best scoring model was Densenet. From 3, we can see that our custom methods, provided really good accuracy scores on the validation set, proving we are on the right track by training the network on our own dataset, instead of using pretrained data.

Metric	**Mobile**	**Dense**	**Res**	**Eff**
Train Loss	0.0009	0.0004	0.0023	0.0011
Train Accuracy	0.9896	0.9964	0.9751	0.9893
Val Loss	0.1318	0.0191	0.1421	0.0871
Val Accuracy	0.8682	0.9708	0.9110	0.9352
Val R^2	0.8259	0.9520	0.8182	0.8543
Val MSE	0.2094	0.0577	0.2187	0.1752

Table 3: Model Performance Comparison. 4 different CNN models. Values are from the 20th epoch (the last one) in the training history. Abbreviations for the model names at the top row, corresponding to the same models as in 2

5.3 Custom Trained With Classifiers

Proceeding with our custom trained networks, the next thing we wanted to do was apply XGBoost to the model, as a final layer to hopefully increase accuracy. The reason for XGBoost was that for this classification task, XGBoost outperformed Random Forests classification. For the XGBoost layer, we kept it simple, with no fine-tuning of parameters. The results of XGBoost are shown in 4. Reviewing table 3 and 4, we can compare

Model	Accuracy	R^2	MSE
Densenet	0.9679	0.9294	0.0896
MobileNetV3	0.9318	0.8601	0.1638
ResNet	0.9466	0.8753	0.1460
EfficientNet	0.9502	0.8832	0.1368

Table 4: Performance metrics for different models

how the classifiers impact the models' prediction scores. Comparing the accuracy scores, MSE's and R scores, we can see that the addition of XGBoost improved MobileNetV3, EfficientNet and Resnet performance. However, it actually made the densenet model slightly worse. Overall, the Densenet model, without any form for boosting performed the best. In the appendix 8, we can see that the plain model has the same f-1 scores, as the XGBoosted model. This implies that there is no need for boosting the densenet model in this case, however the other models benefitted.

5.4 Own CNN Classifier

In Figure 7, the results of training the CNN using a fixed learning rate of $\eta = 0.001$, a batch size of 32, and a fixed number of 20 epochs are shown. With these parameters, a test set accuracy of 97.10% was achieved.

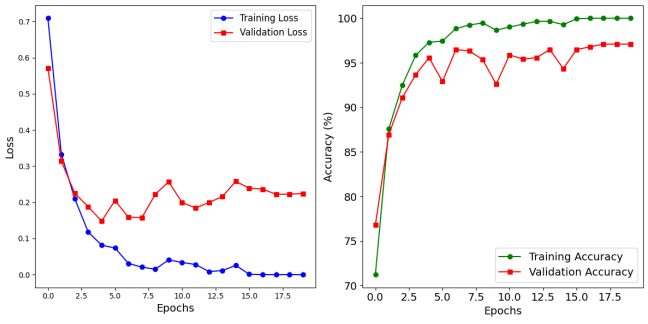


Figure 7: Evolution of the losses and accuracies during the training process.

Additionally, a sweep of the learning rate values from $\eta = 10^{-4}$ to $\eta = 10^{-1}$ was performed, while keeping the batch size fixed at 32 and the number of epochs at 20. The results of this hyperparameter tuning are presented in Figure 8. It can be observed that the optimal value corresponds to $\eta = 0.001$, as smaller values result in slow convergence and larger values lead to divergence. The results of the predictions made by CNN are shown in the confusion matrix in Figure 10(c).

5.5 Principal Component Analysis

To explore alternative classification models such as Logistic Regression, Random Forest, and Feed Forward

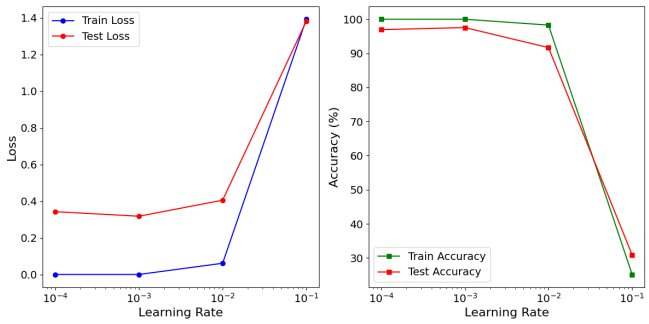


Figure 8: Losses and accuracies obtained as a result of the training using different values of learning rates.

Neural Networks, PCA was first employed to reduce the principal components of the images as much as possible and minimize the dataset's dimensionality. After sweeping through potential numbers of components, the results presented in Figure 9 were obtained.

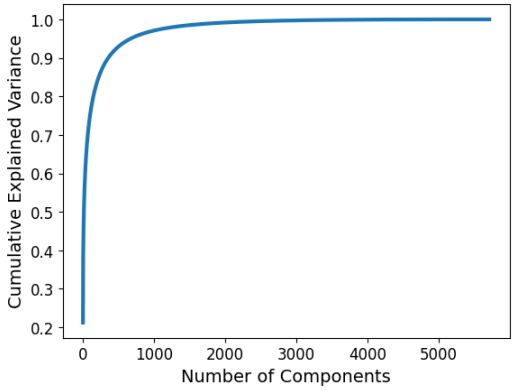


Figure 9: Explained Cumulative Variance as a function of the number of componentes. We can observe that no significant increase is achived after 1000 components.

Based on the elbow method [1] shown in Figure 9, a total of 1000 principal components was selected for the dataset reduction. These newly reduced values were then implemented in the classification models.

A Logistic Regression model from the Scikit Learn library [15] was implemented to classify the four dataset classes. A maximum number of iterations of 1000 was used, and the accuracy metric was applied to the test set to evaluate the algorithm. After the regression was completed, a final accuracy of 88.18% was achieved, and the classification results are shown in Figure 10(a).

Similarly, a Random Forest model from the Scikit Learn library [15] was implemented for the classification task. A maximum of 100 estimators was used, along with a minimum samples split of 2. An arbitrary maximum depth was chosen; therefore, nodes were expanded until all leaves were pure or until all leaves contained fewer than the minimum number of samples required for splitting. The accuracy metric was also used to measure the algorithm's performance. A final accuracy of 89.02% was achieved, and the classification results on the test set are presented in Figure 10(b).

Aditionally, an FFNN was used which takes as input the principal components of the data set and performs a classification of the classes using four output neurons. The network architecture consists of 3 hidden layers with dimensions $[256, 128, 64]$ and whose input layer corresponds to the number of principal components 1000. For the training of this network, the same procedure of the CNNs was followed using an Adam optimizer with a learning rate of 0.001 over a total of 50 epochs. After this training a total accuracy of 94.28% over the test set was obtained. Further details of the training process of this FFNN can be found in `https://github.com/hishemok/FYS-STK4155/tree/main/Project3`

Finally, as a comparison of all these models, Figure 10 shows the confusion matrices obtained by comparing the CNN, Logistic Regression and Random Forest models to see the performance of each model and to observe the specific classes where the classifications fail.

6 Discussion

When comparing the classifiers, Random Forests and Gradient Boosting, we found that Random Forests achieved an accuracy of 84.31%. While it was computationally efficient, it was less effective than Gradient Boosting, which achieved an accuracy of 89.37% with 700 weak learners. However, when using the optimal number of estimators determined from 5-fold cross-validation (20), the accuracy dropped to 84.68%. This suggests that cross-validation may not have provided a reliable estimate of the testing error. Both methods exhibited overfitting, with training accuracies reaching 100%. Future work could focus on improved regularization techniques or alternative pretrained models for feature extraction.

For XGBoost, we explored only a limited set of hyperparameters. A more comprehensive approach, including combining parameters and prioritizing tree-defining parameters before tuning conversion-related ones, could provide better insights and results.

When transitioning to custom-trained CNNs, we observed significantly improved performance, even without classifiers in the final layer. We experimented with multiple models, achieving a peak test accuracy of 97.08%. This highlights the importance of training models on the specific dataset type rather than relying only on pretrained models, which while effective, were outperformed. With additional time, we could have further optimized model parameters to achieve even greater results by adjusting combinations of layers, learning rates, optimizers, and other configurations.

Applying boosting methods to the pretrained models showed mixed effects. For MobileNet, we observed a notable improvement of nearly 6%, whereas for Densenet, it showed minimal or no enhancement. It's worth noting that we did not fine-tune parameters when applying XGBoost, which could have further improved results. MobileNets, in particular, stand out for their efficiency, and when combined with gradient boosting, they can deliver excellent performance in a relatively short time.

Finally, we achieved a test accuracy of 97.10% using a relatively simple custom CNN, slightly outperforming the pretrained DenseNet model. This underscores the potential of custom models when tailored to the specific classification task. Improvements could be made by experimenting with additional layers or configurations, though this would require significant computational resources. However by the looks of our results, you could possibly achieve an incredibly good score by tweaking all possible configurations, until you find the best model for your classification task.

For the PCA method, it did not compare with the trained CNNs. It might be because it is not very effective to represent images as a features vector for training. Some information could be missing and the relation between nearest pixels of the images could be lost.

7 Conclusion

In conculsion, our comparison of classifiers highlighted the strengths and limitations of different approaches for MRI image classifications. While random forests offered computational efficiency, Gradient Boosting achieved better accuracy. XGBoost proved to be a better gradient booster for the classification task. However, the performance could likely be enhanced by a more in depth exploration of hyperparameters.

Custom-trained CNNs proved to be a definetively better approach than using pre-trained models. We found that the custom model, Densenet, achieved a score of 97.08%, with little to no difference made by applying XGBoost. However, our custom made and custom trained model, achieved a slightly better score of 97.10%.
Boosting methods showed mixed effects, with the greatest results, when applied to MobileNets, where we achieved an increase of 6% in accuracy.
The underperformance of PCA shows its limitations in representing image data effectively for classification tasks, as it may lose crucial spatial information. Overall, our finding emphasize the value of custom trained CNNs, robust hyperparameter optimizations, and the integration of boosting methods in classification task. We found that with relatively simple models, we were able to achieve very good results in classification of brain tumors. This shows the value of machine learning in the medical industry. With even more data, and even more time at hand, we are able to create very robust and very acurate models, which could improve efficiency in medical institutions.

References

[1] Mani Bayani. *Robust PCA Synthetic Control*. 2021. arXiv: 2108.12542 [econ.GN]. URL: https://arxiv.org/abs/2108.12542.

[2] Leiyu Chen et al. "Review of Image Classification Algorithms Based on Convolutional Neural Networks". In: *Remote Sensing* 13.22 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13224712. URL: https://www.mdpi.com/2072-4292/13/22/4712.

[3] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, Aug. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[4] Torch Contributors. *PyTorch. Models and pre-trained weights*. 2017. URL: https://pytorch.org/vision/stable/models.html.

[5] scikit-learn developers. *scikit-learn. sklearn.decomposition: PCA*. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[6] xgboost developers. *XGBoost Documentation*. 2022. URL: https://xgboost.readthedocs.io/en/latest/index.html.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009. ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_3.

[8] Morten Hjorth-Jensen. "13. Neural networks". In: *Applied Data Analysis and Machine Learning*. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter9.html.

[9] Morten Hjorth-Jensen. "14. Building a Feed Forward Neural Network". In: *Applied Data Analysis and Machine Learning*. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapter10.html.

[10] Morten Hjorth-Jensen. "7. Optimization, the central part of any Machine Learning algortithm". In: *Applied Data Analysis and Machine Learning*. URL: https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/chapteroptimization.html#steepest-descent.

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. eprint: 1412.6980. URL: https://arxiv.org/abs/1412.6980.

[12] Hishem Kløvnes, Gerlind Deschner, and Juan Manuel Scarpetta. "Project 1; FYS-STK4155 – Applied Data Analysis and Machine Learning". In: (2024).

[13] Hishem Kløvnes, Gerlind Deschner, and Juan Manuel Scarpetta. "Project 2; FYS-STK4155 – Applied Data Analysis and Machine Learning". In: (2024).

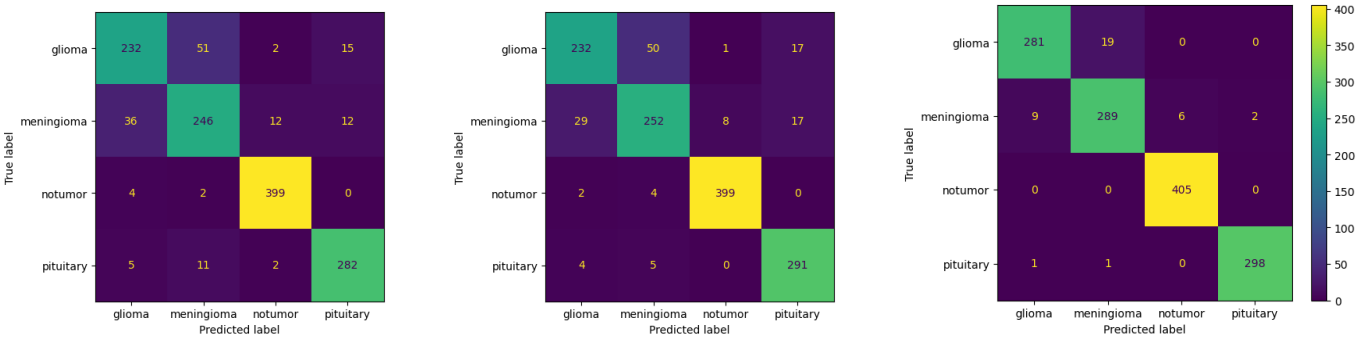
[14] Masoud Nickparvar. *Kaggle. Brain Tumor MRI Dataset: A dataset for classify brain tumors*. 2021. URL: https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset/data.

[15] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[16] Sebastian Raschka, Yuxi Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt Publishing, 2022. ISBN: 91801819319.

8 Appendix

The python code, plots, data and jupyter notebooks can be found in the Project3 folder in the main branch under the following github page https://github.com/hishemok/FYS-STK4155/tree/main/Project3.



(a) Logistic Regression (b) Random Forest (c) Convolutional NN

Figure 10: Confusion matrices evaluated on three different classification models (a) Logistic Regression (b) Random Forest and (c) Convolutional Neural Network.

Class plain	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.84	0.93	0.88	302
Meningioma Tumor	0.87	0.81	0.84	344
No Tumor	0.97	0.96	0.96	410
Pituitary Tumor	0.96	0.94	0.95	348
Accuracy			**0.91**	**1404**
Macro Avg	0.91	0.91	0.91	1404
Weighted Avg	0.91	0.91	0.91	1404

Class XGBoost	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.91	0.90	0.90	302
Meningioma Tumor	0.87	0.88	0.88	344
No Tumor	0.97	0.96	0.97	410
Pituitary Tumor	0.94	0.95	0.95	348
Accuracy			**0.93**	**1404**
Macro Avg	0.92	0.92	0.92	1404
Weighted Avg	0.93	0.93	0.93	1404

Table 5: Classification reports for the two methods: the top table shows results from the MobileNetV3 model without XGBoost, while the bottom table shows results with XGBoost.

Class plain	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.93	0.95	0.94	302
Meningioma Tumor	0.92	0.87	0.90	344
No Tumor	0.99	0.97	0.98	410
Pituitary Tumor	0.93	0.97	0.95	348
Accuracy			**0.94**	**1404**
Macro Avg	0.94	0.94	0.94	1404
Weighted Avg	0.94	0.94	0.94	1404

Class XGBoost	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.96	0.95	0.95	302
Meningioma Tumor	0.91	0.91	0.91	344
No Tumor	0.99	0.96	0.97	410
Pituitary Tumor	0.93	0.96	0.95	348
Accuracy			**0.95**	**1404**
Macro Avg	0.95	0.95	0.95	1404
Weighted Avg	0.95	0.95	0.95	1404

Table 6: Classification reports for the EfficientNet model: the top table shows results from the model without XGBoost, while the bottom table shows results with XGBoost.

Class plain	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.90	0.93	0.92	302
Meningioma Tumor	0.93	0.79	0.85	344
No Tumor	0.91	0.98	0.94	410
Pituitary Tumor	0.94	0.97	0.96	348
Accuracy			**0.92**	**1404**
Macro Avg	0.92	0.92	0.92	1404
Weighted Avg	0.92	0.92	0.92	1404

Class XGBoost	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.91	0.91	0.91	302
Meningioma Tumor	0.90	0.87	0.89	344
No Tumor	0.96	0.98	0.97	410
Pituitary Tumor	0.96	0.97	0.97	348
Accuracy			**0.94**	**1404**
Macro Avg	0.93	0.93	0.93	1404
Weighted Avg	0.94	0.94	0.94	1404

Table 7: Classification reports for the ResNet model: the top table shows results from the model without XGBoost, while the bottom table shows results with XGBoost.

Class plain	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.95	0.97	0.96	302
Meningioma Tumor	0.98	0.93	0.95	344
No Tumor	0.97	0.99	0.98	410
Pituitary Tumor	0.97	0.98	0.98	348
Accuracy			**0.97**	**1404**
Macro Avg	0.97	0.97	0.97	1404
Weighted Avg	0.97	0.97	0.97	1404

Class XGBoost	**Precision**	**Recall**	**F1-Score**	**Support**
Glioma Tumor	0.95	0.96	0.96	302
Meningioma Tumor	0.96	0.94	0.95	344
No Tumor	0.99	0.99	0.99	410
Pituitary Tumor	0.96	0.97	0.97	348
Accuracy			**0.97**	**1404**
Macro Avg	0.97	0.97	0.97	1404
Weighted Avg	0.97	0.97	0.97	1404

Table 8: Classification reports for the DenseNet model: the top table shows results from the model without XGBoost, while the bottom table shows results with XGBoost.