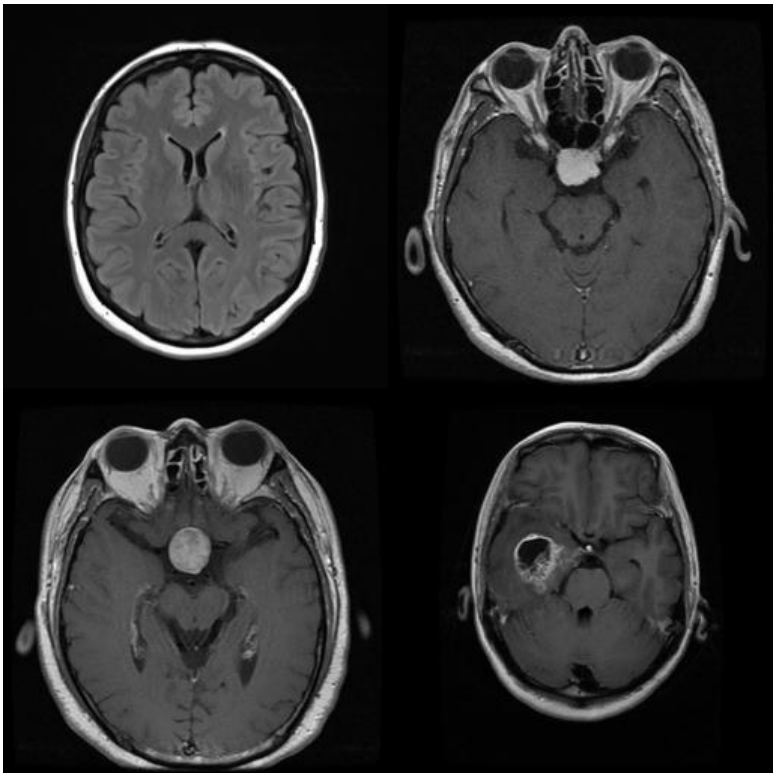

Figure 1: Examples from the dataset [1]

Brain Tumor Classification Using Convolutional Neural Networks

PROJECT REPORT

Anonymous | CS-C3240 - Machine Learning D | 28.3.2021

1. Introduction

Machine aided diagnostics and automated screening of medical data are promising methods in modern healthcare. Nowadays, the extensiveness of medical screening is mostly restricted by shortage of experienced professionals to annotate the data. If computers can be used to at least cut down the amount of expert work, the benefits in both economic efficiency and prevention of severe diseases by early diagnostics can be tremendous.

In this project a simple-structured Convolutional Neural Network, *CNN*, is harnessed to tackle this problem. In the upcoming section 2, I will first discuss the problem formulation more specifically. In section 3 I will then move on to the methods, namely the data and model structure used in this project. I will thereby also define the different model alternatives used. The results of the project are presented in section 4, and section 5 summarizes the work and discusses the limitations and possible issues.

2. Problem Formulation

The aim of this project was to build a machine learning model that can classify brain MRI images into four diagnostical categories:

Healthy	Glioma (tumor)	Meningioma (tumor)	Pituitary tumor

Thus, a dataset containing annotated (labeled) images of each of the four categories was required. Explicitly stated: in this project an image represents a datapoint, it's pixel values are it's features and it's diagnostical category is it's label. However, I must add that in the case of Convolutional Neural Networks the concept of *feature* is not quite unambiguous, while the features can be seen to be learned by the network.

3. Methods

3.1 DATA

The data used in this project was fetched from a private GitHub-repository[1] and consists of altogether 3264 MRI-images of the brain. The distribution of the data between the different classes as well as the division to train and test sets is shown in table 1 below. The data were readily divided to training and testing directories at the source and I chose to keep this division, even though the proportions are slightly inconsistent. In addition to this, 20 % of the training data was split for validation in the model fitting phase.

	Training + validation	Testing	Total
Healthy	316+79	105	500
Glioma	661+165	100	926
Meningioma	658+164	115	937
Pituitary tumor	662+165	74	901
Total	2296+575	394	3264

Table 1: Data distribution

3.2 PREPROCESSING

Before feeding into the model, all the images were reshaped to 256x256 resolution and their pixel values were normalized to a common (0,255) scale. Inside the network, these values would further be converted into (0,1)-scale.

3.3 MODEL STRUCTURE AND CANDIDATE MODELS

The model was constructed using Tensorflow and the Keras Sequential-class in Python. The model structure, adopted from [2] is as follows:

- Rescaling to (0,1)
- 2D Convolution with 16 filters, kernel size 3, even ('same') padding, Relu activation
- 2D Maximum Pooling
- 2D Convolution with 32 filters, kernel size 3, even ('same') padding, Relu activation
- 2D Maximum Pooling
- Flattening
- Dense layer with 128 units, Relu activation
- Dense layer with 4 (number of classes) units (outputs a vector of 4 probabilities)

The exact hypothesis space of a CNN is hard to define in a closed form, but for artificial neural networks generally the value of each node s (including the final node that defines the hypothesis) is a linear combination of the values of (some of) the nodes of the previous layer followed by a (typically non-linear) activation function $g()$(in my case Relu). That is to say, the output of the j:th node is $s_j = g(\sum_{r=1}^{n} w_{j,r} x_r)$[3].

To have alternative models to compare, I chose to use the same network structure but alter some of the hyperparameters, namely the number of training epochs and the size of the training batch. I could have also altered the structure of the network, for example by adding more layers, changing the kernel size or the amount of filters. However, as the simple structure presented above readily seemed to perform quite well, I chose not to change it. I evaluated altogether three models: Model1 having batch size of 32 images and training for 10 epochs, Model2 having batch size of 64 and training for the same 10 epochs and finally Model3 having batch size of 64 but training for 5 epochs only. The choice of the latter ones is further justified in the upcoming section.

The models were trained by running 10/5 epochs with Adam (Adaptive Moment Estimation)[4] -optimizer and using sparse categorical crossentropy as the loss function. This loss function is suitable for mutually exclusive multi-class classification and is readily implemented in the Keras library (keras.losses.SparseCategoricalCrossentropy). Mathematically, average categorical crossentropy can be defined as

$$CE = -\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{M}(y_n)_i \log(\hat{y}_n)_i$$

where N is the number of datapoints, M is the number of labels, y_n is the one-hot encoded true label and $\hat{y}_n$ the corresponding prediction, both vectors of length M, and i represents the label index. Model validation and testing was based on classification accuracy i.e. the number/proportion of correct classifications.

4. Results

4.1 TRAINING

As the models were fitted running 10 or 5 separate epochs, the validation accuracy could be assessed after each epoch separately (using the validation set). In figure 2 below, the

evolution of both training and validation accuracies and losses across the epochs are shown for Model1.

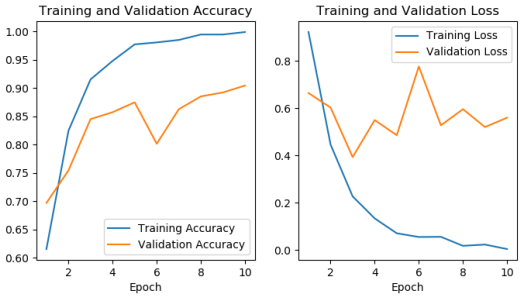

Figure 2: Evolution of training and validation accuracies and losses across the epochs. Model 1

As it is easy to observe from the graphs, the training loss keeps getting lower, and correspondingly the accuracy higher epoch by epoch. The validation accuracy however, seems to saturate much faster, and even though it, too, reaches it's highest value (~90 %) after the whole 10 epochs, there is a huge probability of overfitting. This is prominent based on the loss curves: training loss approaches zero, but validation loss shows no improvement after the first three epochs – rather the opposite – resulting in the approximate value of 0.6.

To experiment with the effect of batch size, I chose to increase it to 64 for the second model. As can be observed from figure 3 below, the validation loss seems to behave somewhat more consistently than in the case of Model1. However, there is still no improvement in validation loss after the first few epochs.

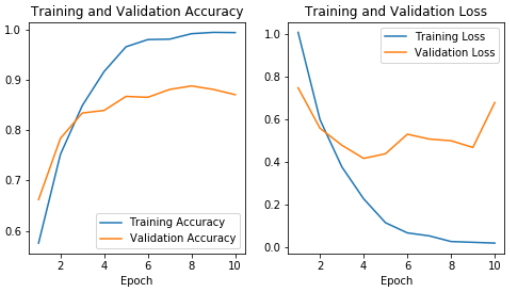

Figure 3: Evolution of training and validation accuracies and losses across the epochs. Model 2

Based on this information I chose to fit yet a third model, that would be essentially the same as the second one, but now only train for 5 epochs. As figure 4 shows, within this timeframe, the losses and accuracies between training and testing seem to remain more coherent.

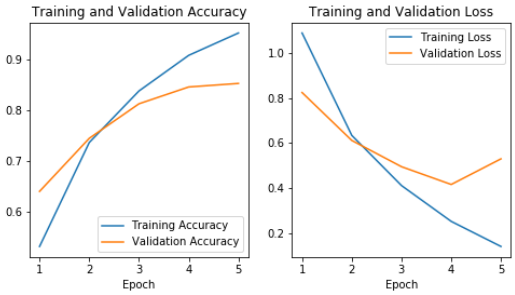

Figure 4: Evolution of training and validation accuracies and losses across the epochs. Model 3

4.2 TESTING

Based on the information presented above, Model3 would be the most appropriate choice to proceed further with. However, I wanted to still evaluate performance of all

three models on the separate test set. In figure 5 below, the resulting accuracies are shown in form of confusion matrices. In figure 6 underneath, the proportional accuracies for each model and class are combined.

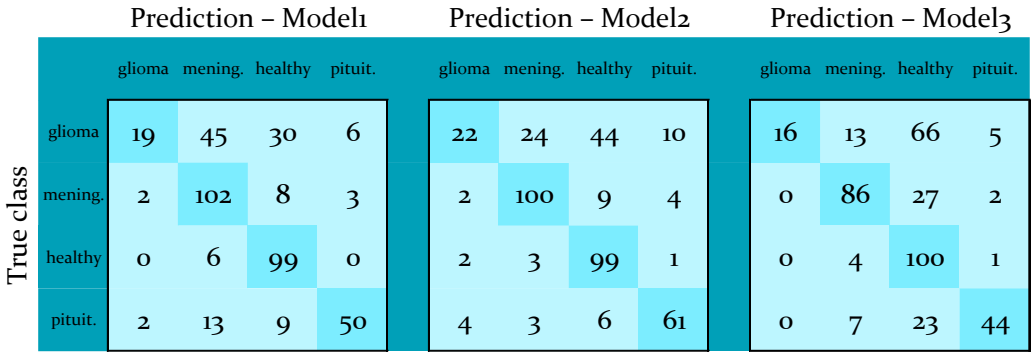

Figure 5: Confusion matrices for each of the three models

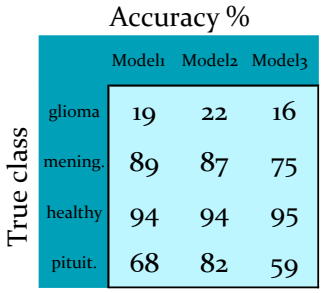

Figure 6: Prediction accuracy as percentage of correct classifications

As the figures show, Model2 has the best overall performance on the test set. Out of the four classes, the glioma-images seem to be the hardest ones for the models to classify correctly. Healthy images, on the other hand, are separated from the tumor-containing-ones with ~95 % accuracy by all three models. The probable cause of this will be further discussed in the next section.

5. Conclusion

Apart from the poor accuracy of glioma classification, the results obtained are surprisingly good considering the simplicity of the used model structure. However, this is most probably due to the quality of the used dataset. The availability of medical imaging data is very poor, especially concerning readily annotated images. The dataset used in this project is acknowledged to be of rather poor quality, and thus the results are to be interpreted with great caution. The main problem with the dataset is that there are other qualities than the ones of interest that vary between the data-subsets of different classes. For example, many of the images that represent the tumor-classes have a sagittal or coronal orientation whereas almost all healthy images are axial. This makes it easy for the network to distinguish between healthy and non-healthy images. There is also a lot of variation in resolution and overall image quality in the raw data. Also, as the data source is somewhat unreliable (no documentation etc.) and as I am no medical expert, it is impossible to tell if the annotation (labeling) of the images is fully accurate.

Achieving close to 100 % separation accuracy of healthy images on real life data would have huge benefits in medical screening as the experts could focus on evaluating only the possibly pathological material. However, as promising as my results may seem, I must conclude that in order to evaluate the model's performance in real life, a much higher quality dataset would be needed.

References

[1] Data Source: GitHub-user Sartaj Bhuvaji.
Brain-Tumor-Classification-DataSet. [Cited on 28.3.2021]. Available from:
https://github.com/SartajBhuvaji/Brain-Tumor-Classification-DataSet

[2] Google Tensorflow. Tensorflow tutorials – Image classification. [online]. [Cited on 28.3.2021]. Available from: https://www.tensorflow.org/tutorials/images/classification

[3] Jung. A. (2021) Machine Learning. The Basics. [online] Available from:
https://github.com/alexjungaalto/MachineLearningTheBasics/blob/master/MLBasicsBook.pdf

[4] Kingma, D.P. & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.