DL Midterm Assignment: Brain Tumor

Course - TI3155TU

Authors: Yan Tavares - 5559790

1. Methodology

1.1 Data loading, preprocessing, batch creation

The data consists of 2764 Magnetic Resonance Imaging images, they are labelled in a ordered manner in a JSON file. Most images have resolution of 512 x 512, but some deviate significantly from this value. Not all the images are oriented as profile, some are from above and some seem to be originated by a cross section of the head.

To process the data into proper inputs and outputs for the convolutional network, it is necessary to create X tensors with shape (batch, channels, height, width) and Y tensors with shape (outputs). The dataframe will initially have a shape of [2764 rows x 2 columns] with class occurrences showed in Table 2. One hot encoding is applied to switch representation from class names to label to vectors and then the data is shuffled using random seed 42.

Table 1. Distribution of labels in dataframe

Tumor Type	Meningioma	Glioma	Pituitary
Meningioma	937	926	901

The jpeg image files are transformed to image objects by means of PIL package. They are resized to 490x490 since it is the average resolution. Because the resonance imaging is fully black and white we can use the conversion 'L' which formats the image object in to only two dimensions (height,width). The image objects are transformed to numpy arrays (that has direct compatibility), normalized, transformed to tensors and inserted in a channel dimension, and inserted to the batch dimension of X.

Since the training is intended to be completed quickly on a regular laptop with GPU accessibility and limited memory, mini-batches method is implemented. There are 100 batches, each containing around 27 x, y tensors. They are saved separated in the data folder to avoid overload of the computer RAM.

1.2 Convolution network structure

The convolutional neural network (CNN) created for this assignment is inspired in the LeNet architecture due to the simplicity of the data. Parameters like kernel_size, stride, padding and hidden_channels can be changed without the need of restructuring the net. There are tree convolutional layers, two pooling layers and two fully connected sigmoid layers. See schematic in Figure 1 which shows the tuned network.

The untuned network hyper-parameters are inspired on the AlexNet. See the snippet for the parameters:

```
{learning_rate = 0.0001, N_in = 1, N_out = 3,
 ↪ H_in = 490, W_in = 490, hidden_channels =
 ↪ [6,16,16], kernels = [3,3,3,3,3], strides
 ↪ = [1,2,1,2,1], paddings = [1,1,1],
 ↪ hidden_layers = [128,128]}
```

"N_in" and "N_out" refers to the number of input channels and output classes. "W_in" x "H_in" is the image input resolution. "Hidden_channels" stores the number of channels that are generated after each convolution. The "kernel" and "strides" are self explanatory. "Paddings" are only applied for the convolution layer, that is why they only contain three elements. "Hidden_layers" stores the number of perceptrons in the fully connected layers. They can be seen bellow:

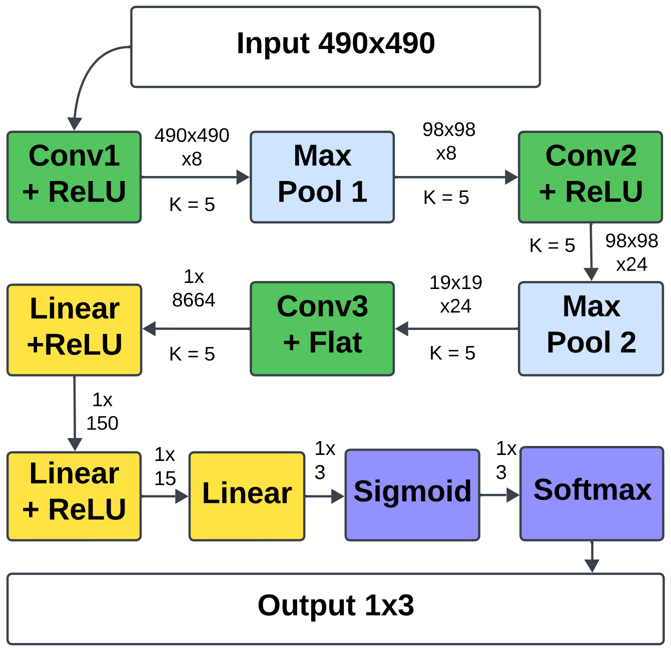

Figure 1. Schematic of the tuned convolutional network

1.3 Network training

Before start training, the stored batches are separated, batches 70 to 100 are unseen (for test), 50 to 69 are validation and 0 to 49 are used for training. During training, forward and backward passes are computed for each batch individually. The loss criteria is cross entropy and it is monitored for the validation and the training sets. The parameters updates are performed by Adam optimizer and the initial parameter are set be means of "xavier_uniform_" function.

Early stopping can be used and it will store two models, one model will be stored for the last local minima detected and the second is stored when the average gradient in the previous 5 epochs are positive. The tuning is done by means of trial and error, different random seeds are also tried and the learning rate is kept as 0.001.

1.4 Evaluation

2. Results

2.1 Dummy classifier

Since there is a very similar distribution of data for each type of cancer, a dummy classifier that randomly assigns one of the three classes would be approximately 33.33%. A classifier like this can can provide a absolute minimum base ground for quality.

2.2 Untuned net

The trained untuned CNN gives a **Accuracy of 81,8%** and a **F1 score of 82,1%**.

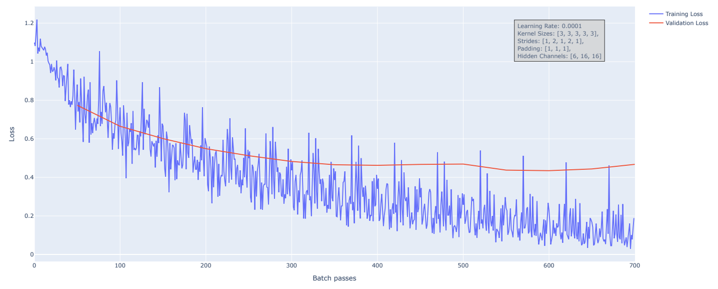

Figure 2. Loss plot of untuned model

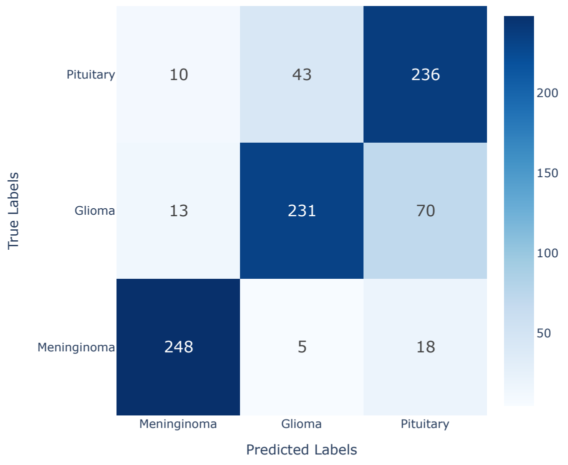

Figure 3. Confusion matrix of tuned model

2.3 Tuned net

After numerous tests, it has been noted that bigger kernels like 5x5 and 7x7 are more suitable for this small CNN structure and input picture size of 490 x490. The increase in the number of channels helped, however increasing much more than 24 did not seem to improve performance. The tests demonstrated to be highly sensitive to weight initialization, and therefore has a high variation. Random seed = 0 led to satisfactory results.

The trained and tuned CNN gives a **Accuracy of 86,3%** and a **F1 score of 86.4%**.

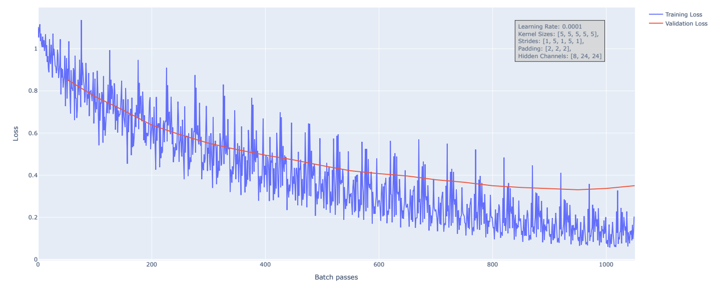

Figure 4. Loss plot of tuned model

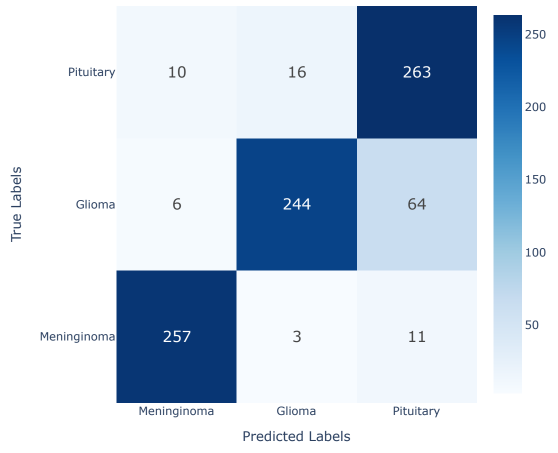

Figure 5. Confusion matrix of tuned model

3. Conclusion

Table 2. Distribution of labels in dataframe

	Accuracy	**F1 Score**	**Kernels**	**Chanels**	**Seed**
Untuned	81,8	82,1	[3,3,3,3,3]	[6,16,16]	42
Tuned	87.4	87.6	[5,5,5,5,5]	[8,24,24]	0

The CNN evaluated in this report has high overall accuracy, but low accuracy when identifying the difference between Glioma and Pituitary. Approximately 15% of all Glioma cases are wrongly predicted as Pituitary and 8% of Pituitary are wrongly predicted as Glioma.

The CNN showed high variability, requiring tuning of weight initialization. It is speculated that the lack of two sequential convolutions makes it harder for the network to learn directly spacial dependent parameters like stripes and crosses. The loss gradient field seems to have several wide local minima around loss = 3.8 to 0.45 making the training get easily stuck. Therefore with this structure it is unlikely to overpass the accuracy obtained.

It is a quick model, the training takes one to three minutes using a NIVIDIA GeForce RTX 3050 Ti Laptop GPU. For better accuracy, the addition of more convolutional layers is recommended. For more stability consider increasing the batch size.