Brain tumor data set

Alex B. Hemmingsen
Asger D. Christensen
Daniel M. Sørensen
Johan E. Hansen

University of Copenhagen

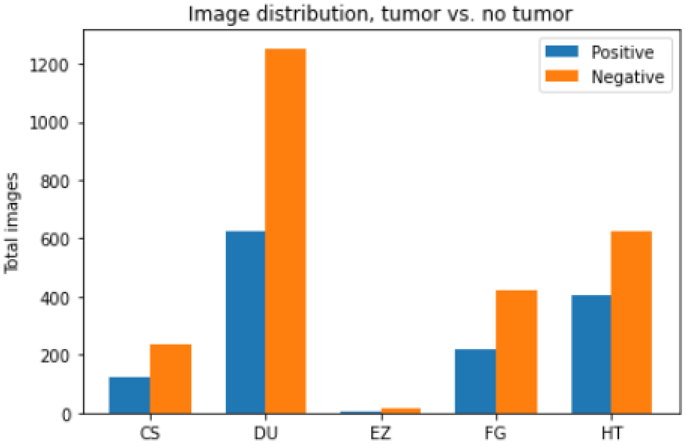
16 June 2021

Overview

Dataset overview

- 110 Patients
  - All patients have been diagnosed with a tumor
  - 3929 images total
- 5 different hospitals
- 20-80 slices from bottom to top of skull
  - RGB Images with dimensions (256,256,3)
  - Masks manually segmented from FLAIR MRI scans
- Link to data: https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation

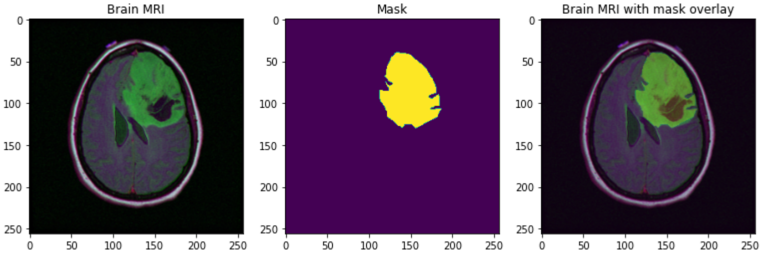
Distribution



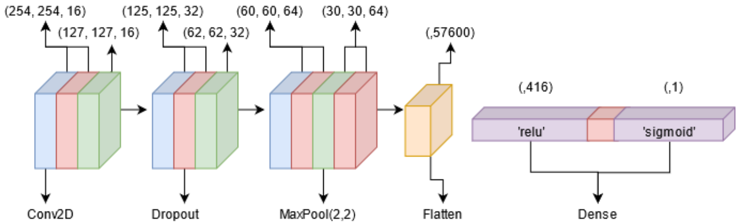
Brain tumor data set

Alex B. Hemmingsen
Asger D. Christensen
Daniel M. Sørensen
Johan E. Hansen

Scan + mask

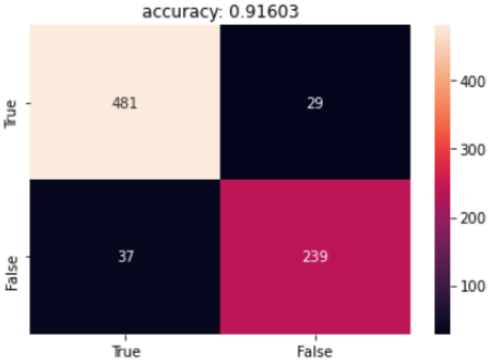


CNN model



Dropout					
0.45	0.25	0.2	0.4		0.15
Activation					
'relu'	'relu'	'relu'			

CNN Results



	Loss	Accuracy
Training set	0.006796	0.998806
Validation set	0.204163	0.918918
Test set	0.295603	0.916030

CNN Optimisation

RandomSearch:

- Dropout value
- Activation layer
- Learning rate

```
model.add(
    Conv2D(
        filters=16,
        kernel_size=3,
        activation='relu',
        input_shape=self.input_shape
    )
)
model.add(
    Dropout(rate=hp.Float(
        'dropout_1',
        min_value=0.0,
        max_value=0.5,
        default=0.15,
        step=0.05,
    ))
)
model.add(MaxPooling2D(pool_size=2))
```

CNN Performance

RandomSearch:

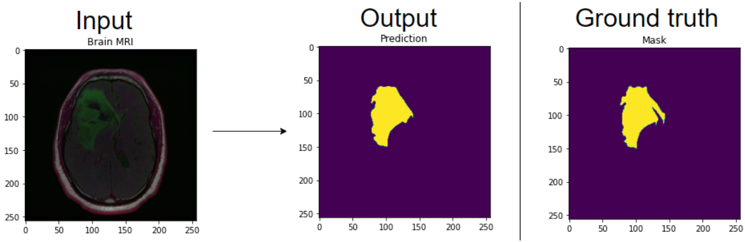
- 20 trials
- 20 epochs per hypermodel
- 2 executions per trial

Training with 2514 images and validating with 629 images, takes ~ 1hr30min for a GTX 970 with 4gb to perform the random search.

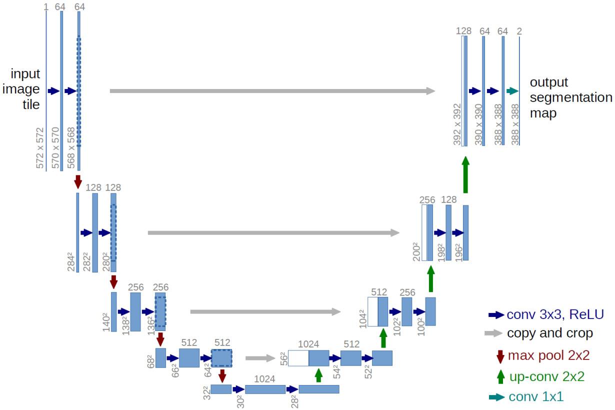
Consider:

- Size of the model
- Number of trials required to saturate extensive number of parameters.

Segmentation



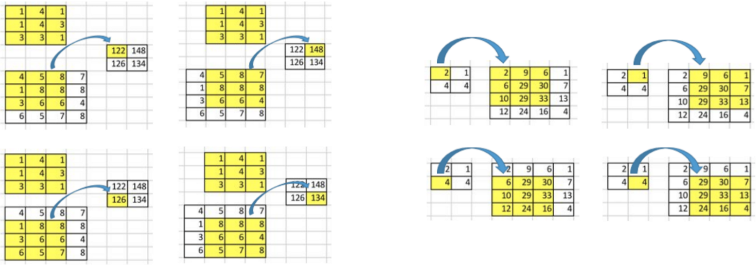
UNet



Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015.

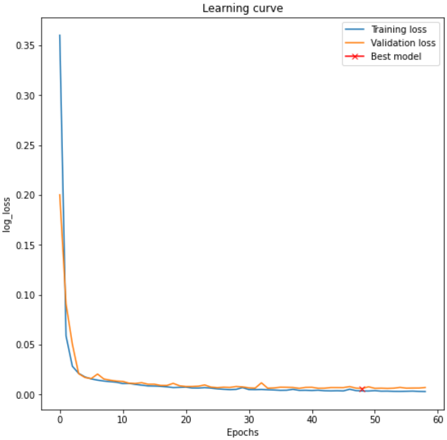
Transposed convolution

Explanation of the transposed convolution
with a 4x4 input, a 3x3 kernel, no padding and stride 1



https://naokishibuya.medium.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

Unet Training



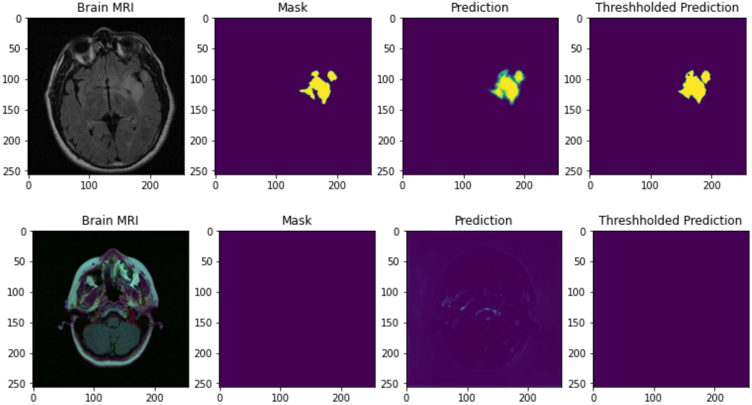
Training with 2357 and validating with 786 images, takes $\sim$ 18min for a RTX 2080 Super 8gb.

Unet evaluation

	Loss	Accuracy
Training set	0.002845	0.998796
Validation set	0.007135	0.997800
Test set	0.008624	0.997651

Is accuracy a good metric in this case? not really.

UNet output

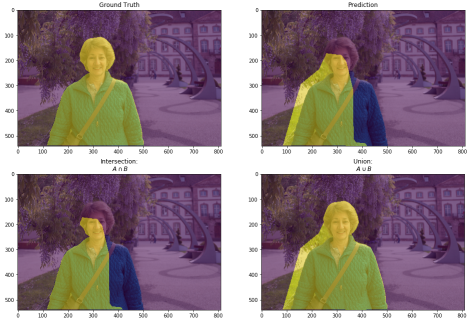


Using threshold > 0.5

IoU score

The IoU equation

$$IoU = \frac{target \cap prediction}{target \cup prediction}$$



https://www.jeremyjordan.me/evaluating-image-segmentation-models/

IoU results

	IoU score
Training set	0.8901
Validation set	0.8135
Test set	0.7918

Image segmentation results (Good)*

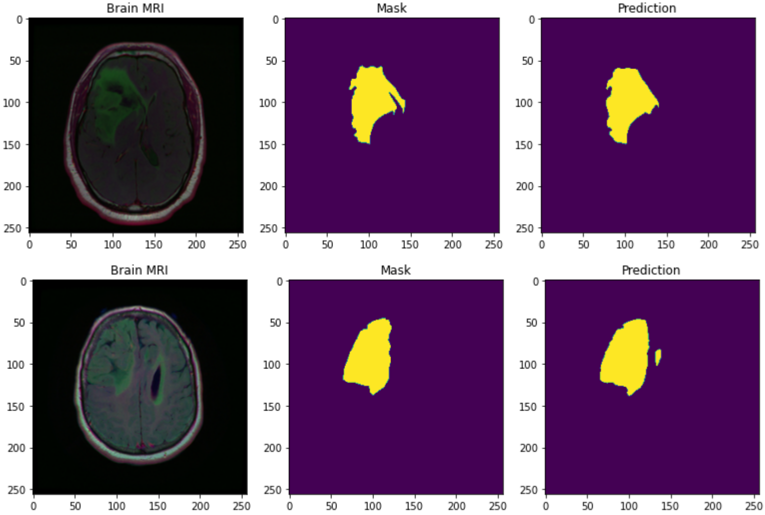
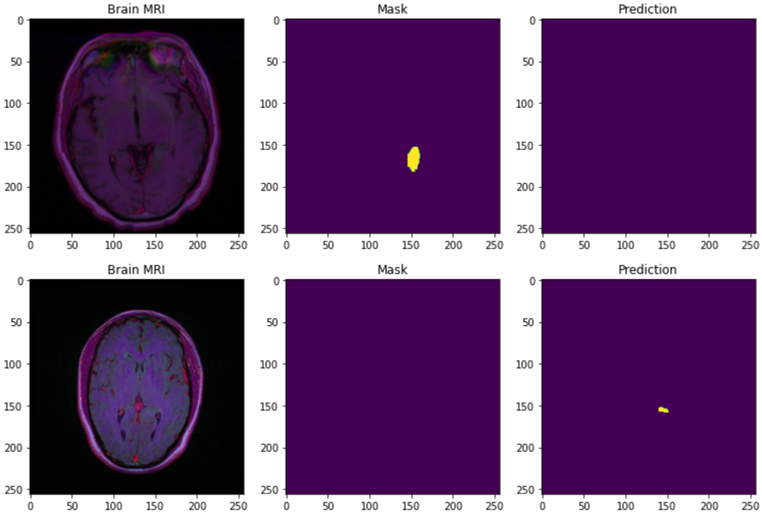


Image segmentation results (Bad)



Considerations

Data augmentation

- Not needed (Model is already alright)
- Might confuse this highly specialized model

Thresholding

- Test different thresholding values

Additional analysis

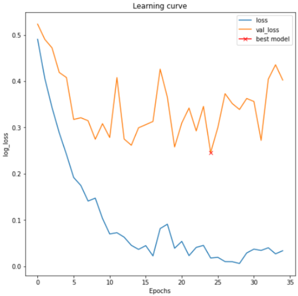
- Is the model more precise at MRI from specific hospitals?
- Is the model more precise at specific layers of the brain?

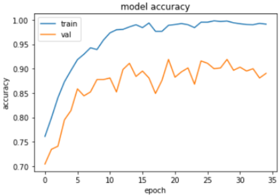
Appendix - CNN RandomSearch

RandomSearch 10 best trials									
drop_1	drop_2	drop_3	drop_4	drop_5	units_1	dense_1	dense_2	learn_rate	val_acc
0.45	0.25	0.2	0.4	0.15	416	relu	sigmoid	0.001027	0.897456
0.3	0.1	0.15	0.25	0.25	480	sigmoid	sigmoid	0.891891	0.891891
0.15	0.2	0.4	0.25	0.2	64	relu	sigmoid	0.001329	0.888712
0.25	0.05	0.5	0.45	0.45	320	relu	sigmoid	0.001486	0.887917
0.05	0.25	0.35	0.1	0.2	416	relu	sigmoid	0.000145	0.885532
0.05	0.25	0.05	0.0	0.35	128	relu	sigmoid	0.000324	0.885532
0.05	0.05	0.35	0.35	0.05	480	sigmoid	sigmoid	0.000362	0.883147
0.1	0.05	0.2	0.45	0.2	288	relu	sigmoid	0.000244	0.881558
0.1	0.3	0.25	0.3	0.05	352	sigmoid	sigmoid	0.001087	0.880763
0.35	0.1	0.05	0.15	0.25	480	relu	sigmoid	0.000115	0.871224

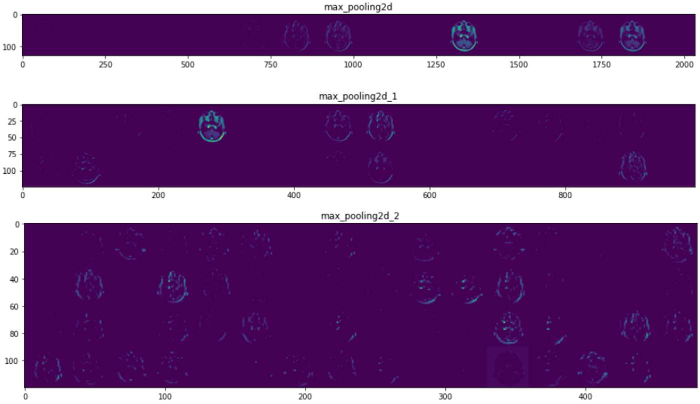
Appendix - CNN training

Graph showing the learning curve and model accuracy when training on the model, which was picked using RandomSearch.





Appendix - CNN MaxPooling layer images



Appendix - Transposed Convolution Calculations

	0	1	2
0	1	4	1
1	1	4	3
2	3	3	1

Kernel (3, 3)

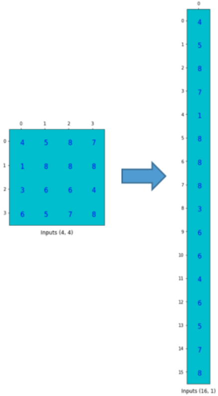
We rearrange the 3x3 kernel into a 4x16 matrix as below:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0	0
1	0	1	4	1	0	1	4	3	0	3	3	1	0	0	0	0
2	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1	0
3	0	0	0	0	0	1	4	1	0	1	4	3	0	3	3	1

Convolution Matrix (4, 16)

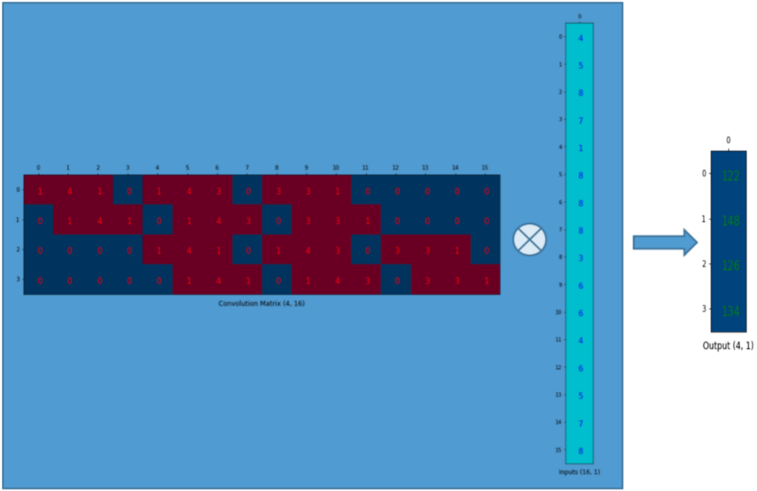
Appendix - Transposed Convolution Calculations

To use it, we flatten the input matrix (4x4) into a column vector (16x1).

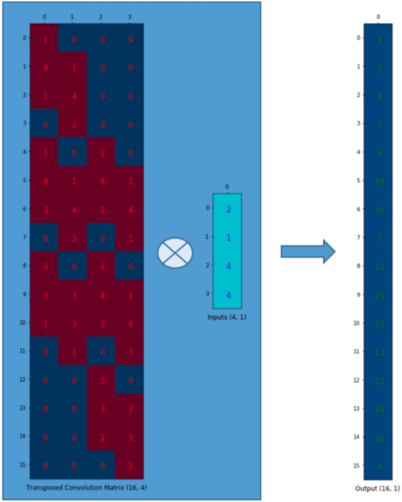


Flattened Input Matrix

Appendix - Transposed Convolution Calculations

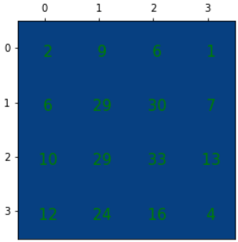


Appendix - Transposed Convolution Calculations



Appendix - Transposed Convolution Calculations

The output can be reshaped into 4x4.



Output (4, 4)