**PURBANCHAL UNIVERSITY**

**FACULTY OF SCIENCE AND TECHNOLOGY**

**COLLEGE OF BIOMEDICAL ENGINEERING AND APPLIED SCIENCES**

Hadigaun, Kathmandu



Report on

Brain Tumor Multi-Classification

Submitted By:  Submitted To:

Kushal Devkota – BME/2017/A26 Shreeya Khadka

Date of Submission

6th May, 2022

Table of Contents

[Phase I 1](#_heading=h.gjdgxs)

[Phase II 12](#_heading=h.30j0zll)

# **Phase I**

1. **Convert Images to 2D numpy array and back.**

Different methods to convert image into numpy array and back to image were used.

1. Using matplotlib
2. Using Keras
3. Using OpenCV

In above methods, an image was converted into a numpy array using imread for matplotlib and OpenCV and image\_to\_array for Keras.



Figure 1:Original Image

* Image to numpy array

array([[[ 80., 72., 69.],

[ 79., 69., 67.],

[ 82., 72., 70.],

...,

[ 38., 39., 44.],

[ 40., 41., 46.],

[ 38., 39., 44.]],

...,

[ 69., 70., 75.],

[ 67., 66., 72.],

[ 78., 77., 82.]]], dtype=float32)



Figure 2: numpy array to original image

1. **Set up a basic CNN network to classify different tumor types.**

Basic CNN network was defined using conv2D, maxpooling2D, Dense, flatten layers available in tensorflow package, keras library.

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 198, 198, 32) 896

max\_pooling2d

(MaxPooling (None, 99, 99, 32) 0

2D)

dense(Dense) (None, 99, 99, 128) 4224

flatten(Flatten) (None, 1254528) 0

dense(Dense) (None, 4) 5018116

=================================================================

Total params: 5,023,236

Trainable params: 5,023,236

Non-trainable params: 0

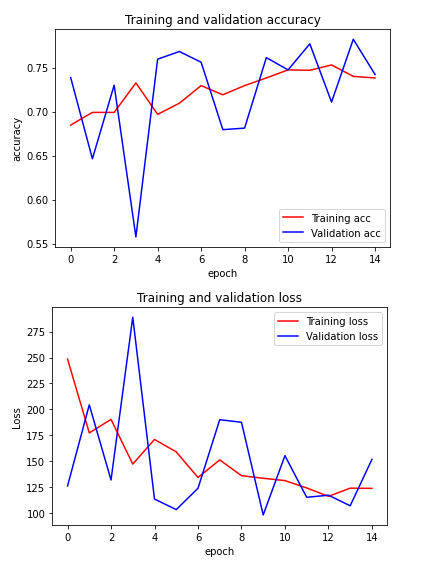
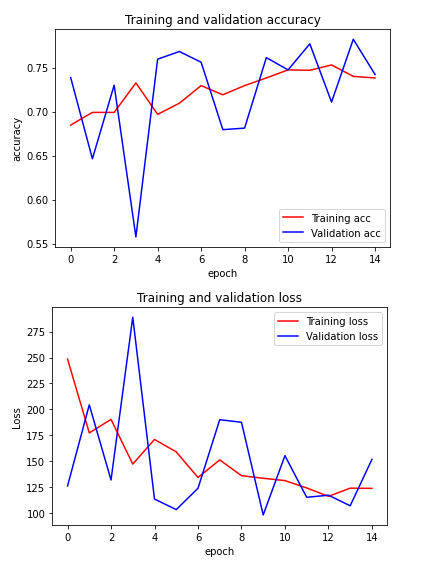
In this base model, sparse categorical cross entropy is used in order to classify tumor types. Adam optimizer is used which adjusts learning rate itself and helps in convergence.

1. **Report key metrics to quantify the performance of your neural network.**

Outcome of base\_model:

loss: 123.9618 – acc: 0.7382

val\_loss: 151.8185 - val\_acc: 0.7422



Confusion matrices for validation dataset:

[[114 13 6 15]

[ 52 79 21 27]

[ 3 1 67 7]

[ 3 0 0 166]]

Confusion matrices for training dataset:

[[27 9 60 4]

[17 34 60 4]

[19 0 86 0]

[14 5 14 41]]

Classification metrices for testing dataset:

precision recall f1-score support

glioma\_tumor 0.35 0.27 0.31 100

meningioma\_tumor 0.71 0.30 0.42 115

no\_tumor 0.39 0.82 0.53 105

pituitary\_tumor 0.84 0.55 0.67 74

accuracy 0.48 394

macro avg 0.57 0.48 0.48 394

weighted avg 0.56 0.48 0.47 394

accuracy: 0.4772 f1\_score: 0.4772

1. **Set up a hyperparameter tuning mechanism to improve your performance metrics. Report performance on multiple scenarios tested. Different number of neurons/filter sizes?**

Hyperparameters tuning used are:

1) Model hyperparameters:

layers of CNN like Conv2D, pooling layers, dense layers and dropout layers, etc.

As model hyperparameters increase the model complexity also increases and time required to complete epoch also increases.

2) Optimizer hyperparameters:

SGD and Adam optimizers were used. Keeping all parameters constant, losses due to SGD was greater as compared to Adam. So, Adam optimizer was preferred as it adjusts the learning rate with increasing epoch.

3)Data hyperparameters:

As the dataset contains different image sizes all images were resized to (200, 200) and validation split was 25% of the training dataset.

New base\_model with five conv2D and maxpooling2D layers:

Increasing model complexity since base models lack proper convolution layers to detect features.

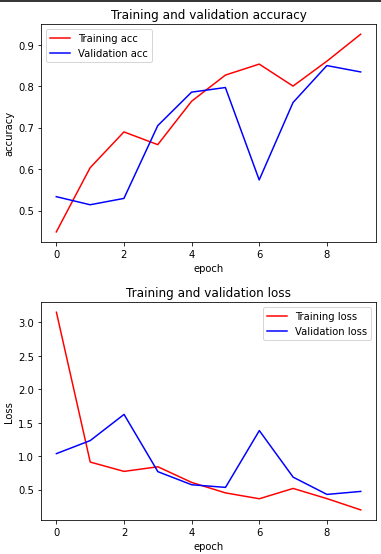
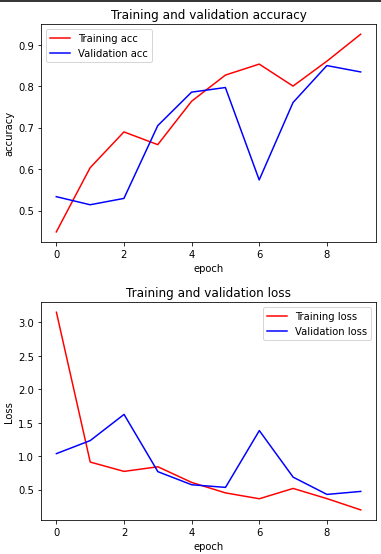
Training this model for:

1. Changing epoch

Changing epoch size helps to understand convergence and helps to detect overfitting.

1. Epoch = 10 ; keeping all parameters same

loss: 0.1968 - acc: 0.9252 val\_loss: 0.4735 - val\_acc: 0.8343



Performance on validation set:

Confusion Matrix

[[167 24 00 01] accuracy: 0.8343

[ 24 165 23 05]

[ 01 7 88 01] f1\_score: 0.8343

[ 5 23 05 179]]

Performance on test set:

Confusion Matrix

[[17 31 51 1] accuracy: 0.5152

[ 3 63 49 0]

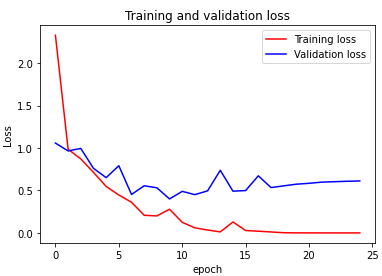
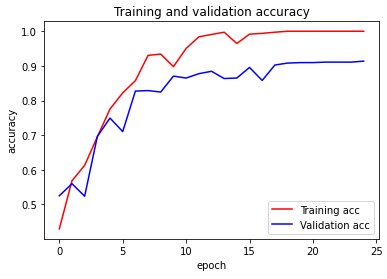
[ 2 5 98 0] f1\_score: 0.5152

[ 2 34 13 25]]

1. Epoch = 25 ; keeping all parameters same

Increasing epoch size to 25 to provide more epoch for convergence.

loss: 1.3191e-04 - acc: 1.0000 val\_loss: 0.6119 - val\_acc: 0.9136



Performance on validation set:

Confusion Matrix

[[178 13 0 1] accuracy: 0.9136 f1\_score: 0.9136

[ 14 187 8 8]

[ 1 11 83 2]

[ 3 1 0 208]]

Performance on test set:

Confusion Matrix

[[ 21 37 38 4] accuracy: 0.6904 f1\_score: 0.6904

[ 1 102 7 5]

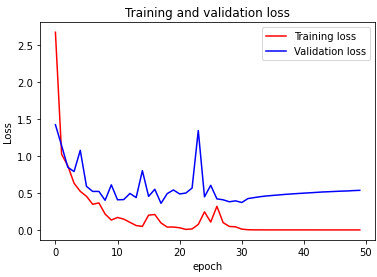
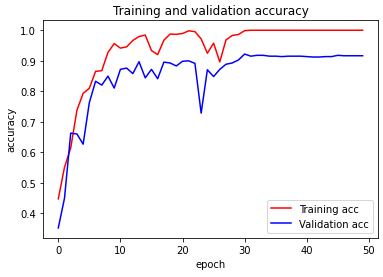
[ 1 6 97 1]

[ 1 15 6 52]]

Increasing epoch size increases the performance on validation and test dataset.

1. Epoch = 50 ; keeping all parameters same

loss: 7.0583e-05 - acc: 1.0000 val\_loss: 0.5349 - val\_acc: 0.9164



Performance on validation set:

Confusion Matrix

[[180 10 0 2] accuracy: 0.9164 f1\_score: 0.9164

[ 14 186 12 5]

[ 1 10 84 2]

[ 1 2 1 208]]

Performance on test set:

Confusion Matrix

[[ 18 37 44 1] accuracy: 0.6624 f1\_score: 0.6624

[ 2 101 10 2]

[ 2 5 97 1]

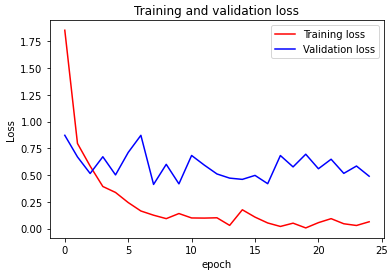
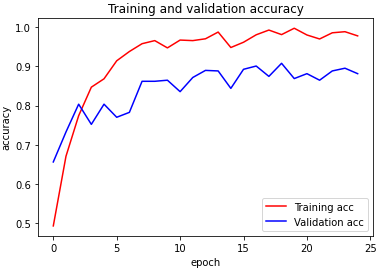
[ 2 14 13 45]]

| No. of Epoch | Accuracy (Validation, test) | F1 score (Validation, test) |
| --- | --- | --- |
| 10 | 0.8343, 0.5152 | 0.8343, 0.5152 |
| 25 | 0.9136, 0.6904 | 0.9136, 0.6904 |
| 50 | 0.9164, 0.6624 | 0.9164, 0.6624 |

1. Changing batch size
2. Batch size = 20

loss: 0.0643 - acc: 0.9782

val\_loss: 0.4896 - val\_acc: 0.8816



Performance on validation set:

Confusion Matrix

[[167 22 0 3] accuracy: 0.8816 f1\_score: 0.8816

[ 19 176 10 12]

[ 1 10 81 5]

[ 2 0 1 209]]

Performance on test set:

Confusion Matrix

[[17 31 46 6] accuracy: 0.6827 f1\_score: 0.6827

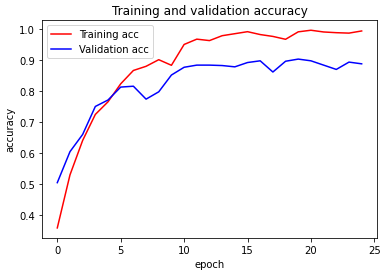
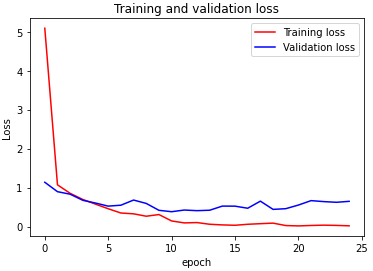
[ 1 99 10 5]

[ 2 6 95 2]

[ 0 6 10 58]]

1. Batch size = 100

loss: 6.4921e-04 - acc: 1.0000 val\_loss: 0.4299 - val\_acc: 0.9164



Performance on validation set:

Confusion Matrix

[[177 14 0 1] accuracy: 0.9164 f1\_score: 0.9164

[ 14 186 8 9]

[ 1 7 87 2]

[ 2 2 0 208]]

Performance on test set:

Confusion Matrix

[[ 22 34 40 4] accuracy: 0.7107 f1\_score: 0.7107

[ 2 103 7 3]

[ 2 5 97 1]

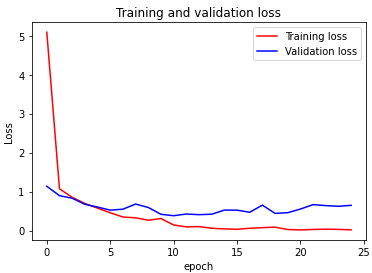
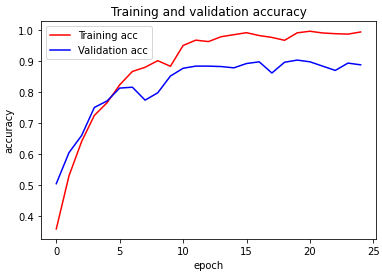
[ 2 12 2 58]]

| Batch size | Accuracy (Validation, test) | F1 score (Validation, test) |
| --- | --- | --- |
| 20 | 0.8816, 0.6827 | 0.8816, 0.6827 |
| 50 | 0.9136, 0.6904 | 0.9136, 0.6904 |
| 100 | 0.9164, 0.7107 | 0.9164, 0.7107 |

1. Changing filter size
2. Filter size = 4\*4

loss: 0.0185 - acc: 0.9949

val\_loss: 0.6466 - val\_acc: 0.8886



Performance on validation set:

Confusion Matrix

[[150 42 0 0] accuracy: 0.8886 f1\_score: 0.8886

[ 7 202 6 2]

[ 0 12 84 1]

[ 1 8 1 202]]

Performance on test set:

Confusion Matrix

[[ 17 52 26 5] accuracy: 0.6675 f1\_score: 0.6675

[ 1 108 4 2]

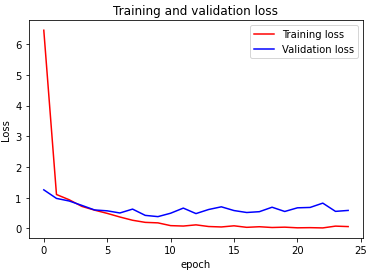
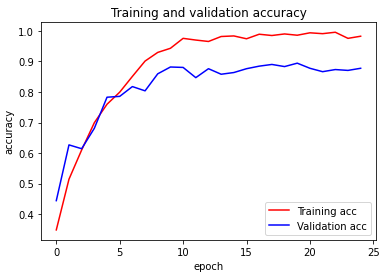
[ 0 6 98 1]

[ 0 34 0 40]]

1. Filter size = 5\*5

loss: 0.0619 - acc: 0.9823

val\_loss: 0.5878 - val\_acc: 0.8774



Performance on validation set:

Confusion Matrix

[[178 13 0 1] accuracy: 0.8774 f1\_score: 0.8774

[ 29 172 12 4]

[ 2 13 76 6]

[ 3 4 1 204]]

Performance on test set:

Confusion Matrix

[[ 19 40 39 2] accuracy: 0.6574 f1\_score: 0.6574

[ 3 102 9 1]

[ 2 6 95 2]

[ 5 18 8 43]]

| Filter size | Accuracy (Validation, test) | F1 score (Validation, test) |
| --- | --- | --- |
| 3\*3 | 0.9136, 0.6904 | 0.9136, 0.6904 |
| 4\*4 | 0.8886, 0.6675 | 0.8886, 0.6675 |
| 5\*5 | 0.8774, 0.6574 | 0.8774, 0.6574 |

1. **Are you overfitting? Verify.**

CNN model performs better on the training set than on the test set, which is likely due to overfitting.

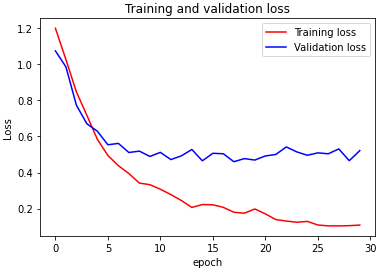
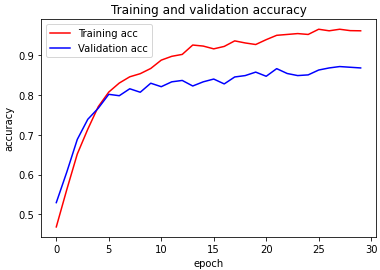
Also, validation and training accuracy isn’t changing which is an indicator for overfitting.

1. **Can you apply additional techniques to improve the performance of your CNN? Dropout?**

Adding dropout layer:

loss: 0.1089 - acc: 0.9608

val\_loss: 0.5214 - val\_acc: 0.8676



Performance on validation set:

Confusion Matrix

[[144 20 0 1] accuracy: 0.8676 f1\_score: 0.867

[ 12 133 10 7]

[ 6 14 47 4]

[ 1 1 0 174]]

Performance on test set:

Confusion Matrix

[[17 34 43 6]

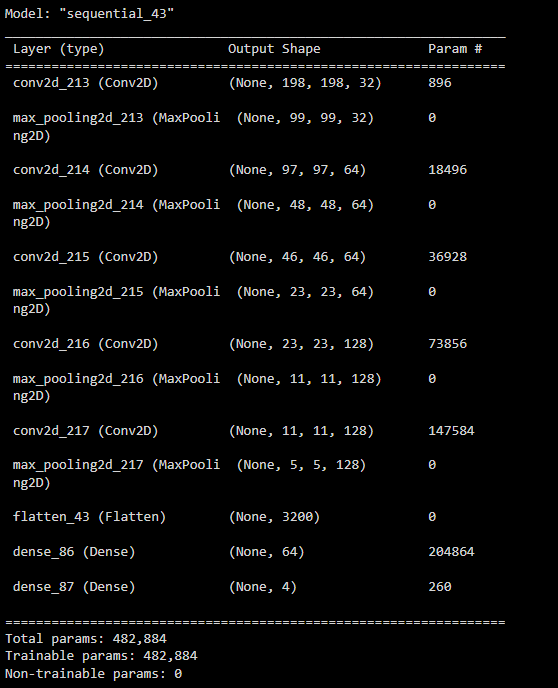
[ 1 98 14 2]

[ 2 8 94 1]

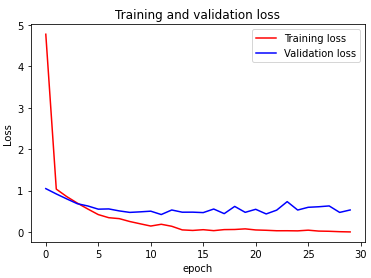
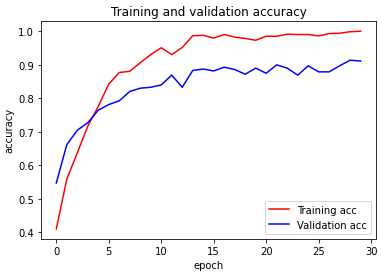
[ 0 17 2 55]]

accuracy: 0.6701 f1\_score: 0.6701

No, significant change was observed. So, best CNN model obtained so far is:



loss: 4.1534e-04 - acc: 1.0000 val\_loss: 0.5310 - val\_acc: 0.9109



Performance on validation set:

Confusion Matrix

[[178 11 0 3]

[ 16 185 7 9]

[ 2 11 82 2]

[ 1 0 2 209]]

accuracy: 0.9109 f1\_score: 0.9109

Performance on test set:

Confusion Matrix

[[ 19 46 31 4]

[ 0 104 8 3]

[ 1 7 97 0]

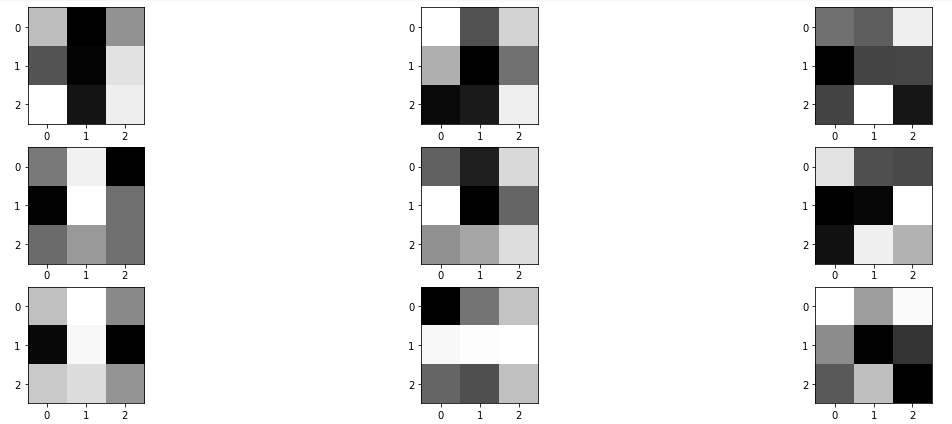
[ 0 11 5 58]]

accuracy: 0.7056 f1\_score: 0.7056

# **Phase II**

### 

1. **Extract filters of your CNN from different layers. Convert them into images. What do they mean?**

****

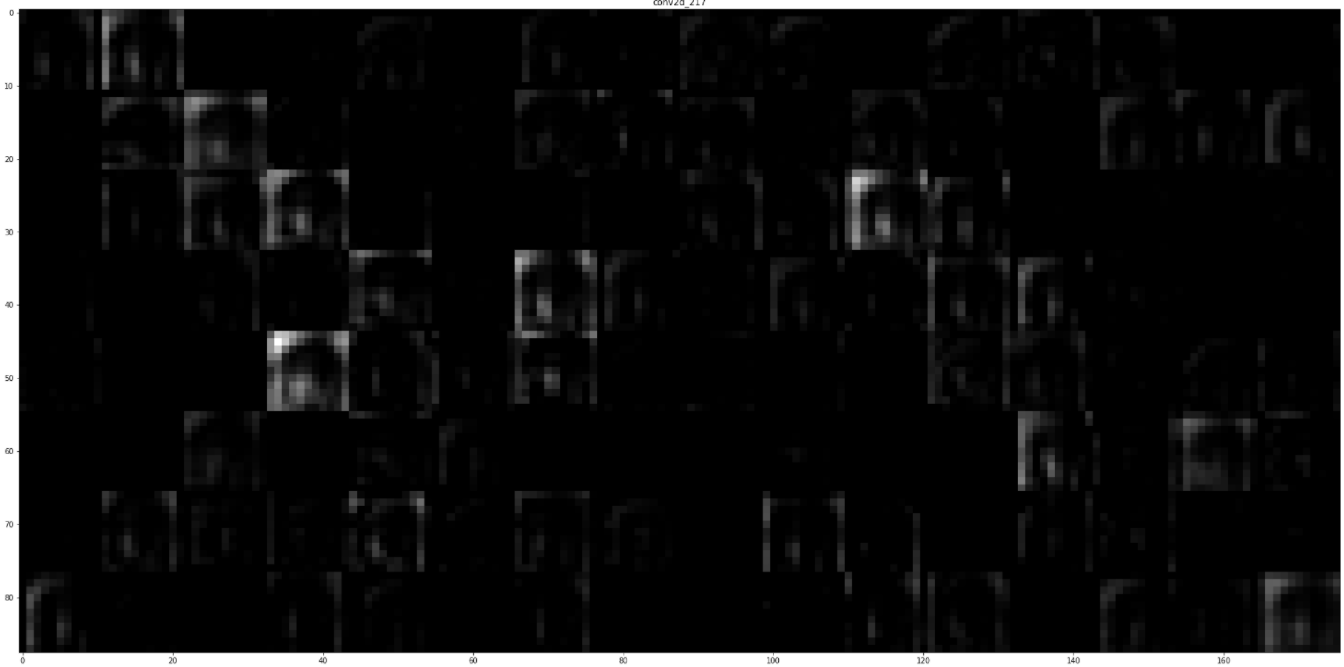
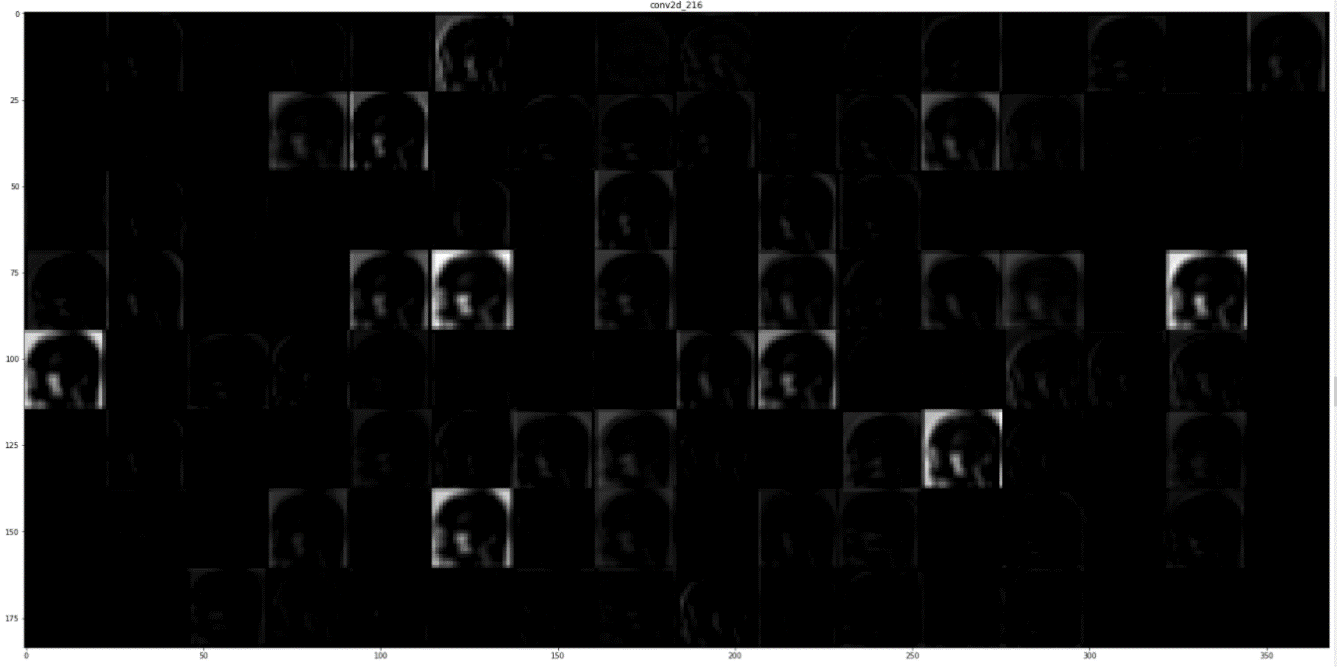
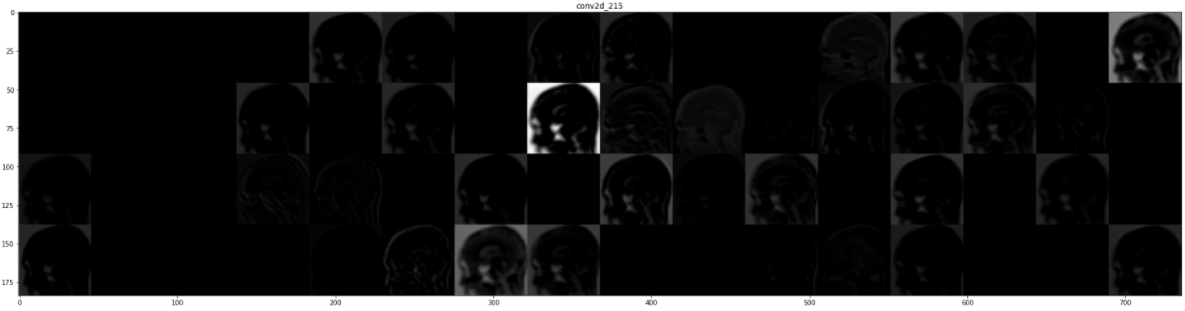
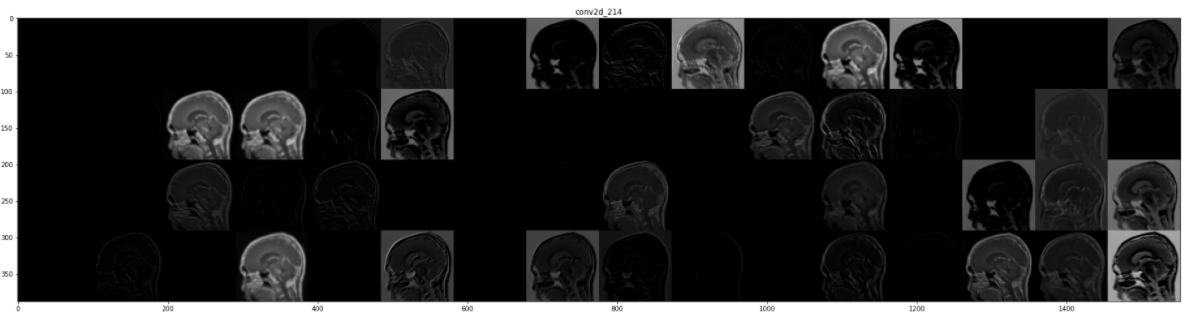
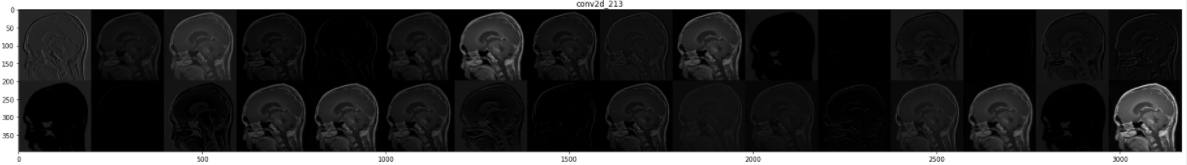
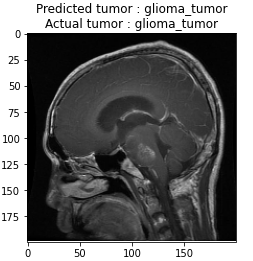
The dark squares indicate small or inhibitory weights and the light squares represent large or excitatory weights. Based on this, features from different convolution layers are extracted.

The above filters are 3 filters out of 32 filters of the first convolution layer.

If we wish to start looking at filters in the second convolutional layer, we have 64 filters, but each has 32 channels to match the input feature maps. To see all 32 channels in a row for all 64 filters would require (32×64) 2048 subplots in which it may be challenging to see any detail.

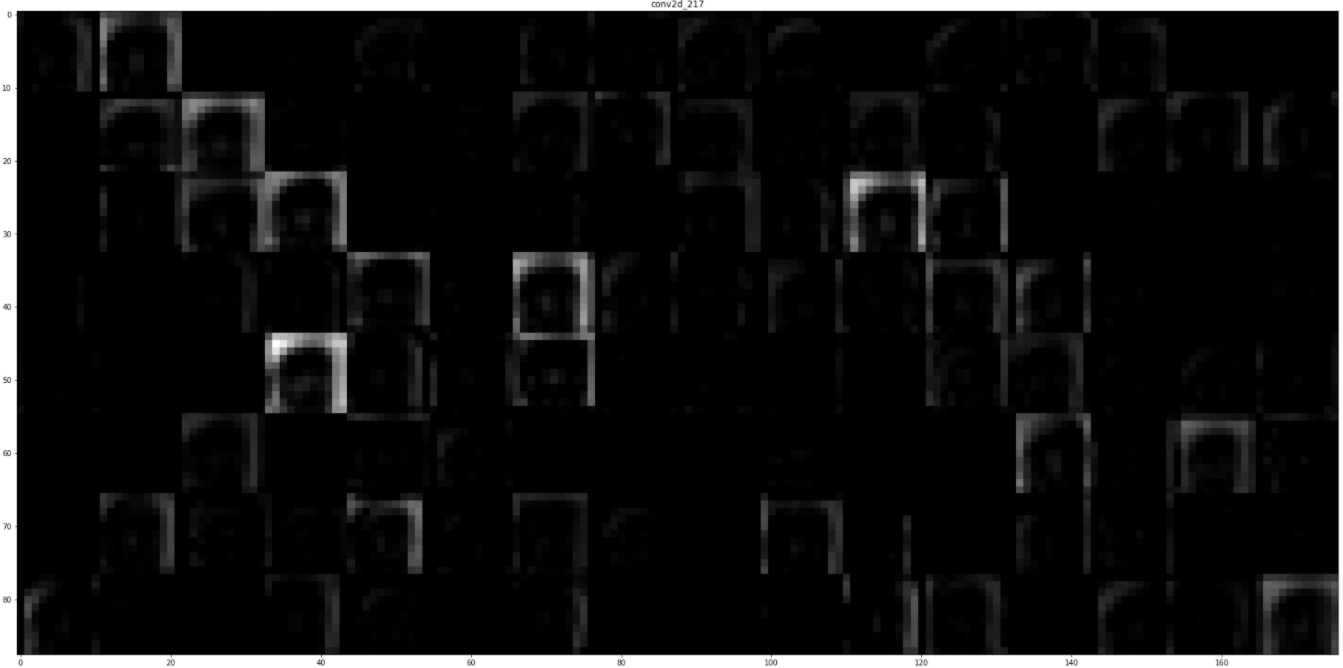
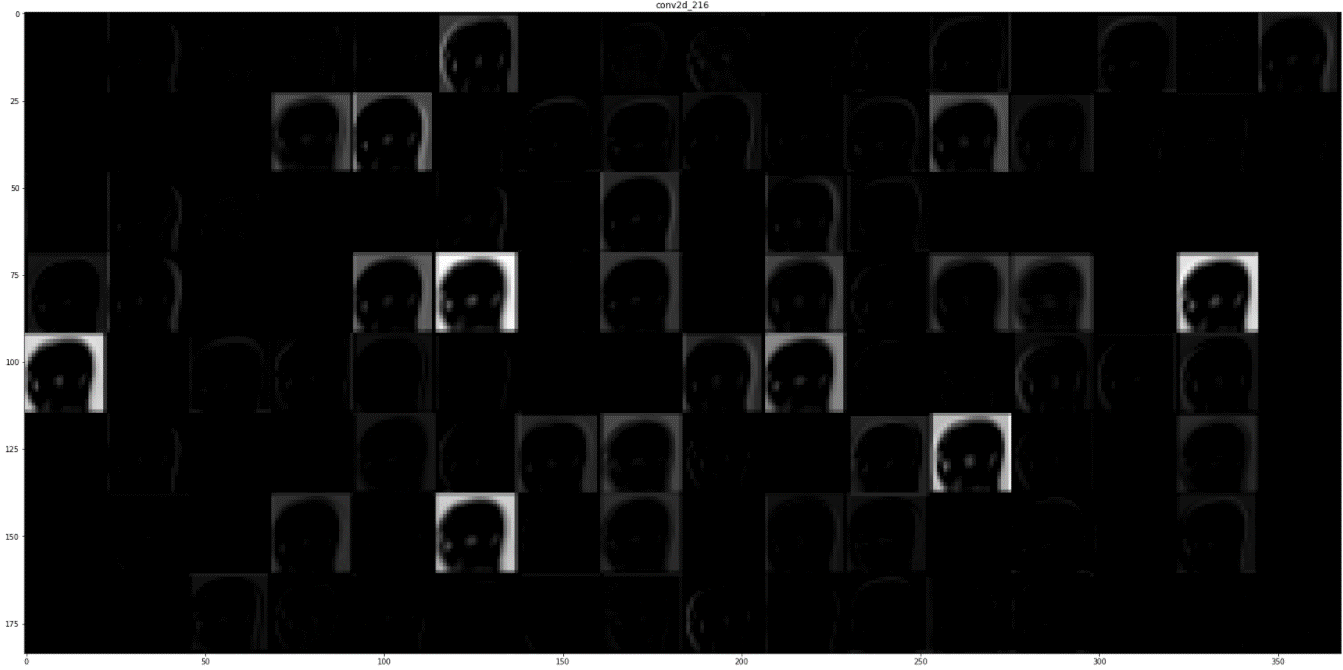
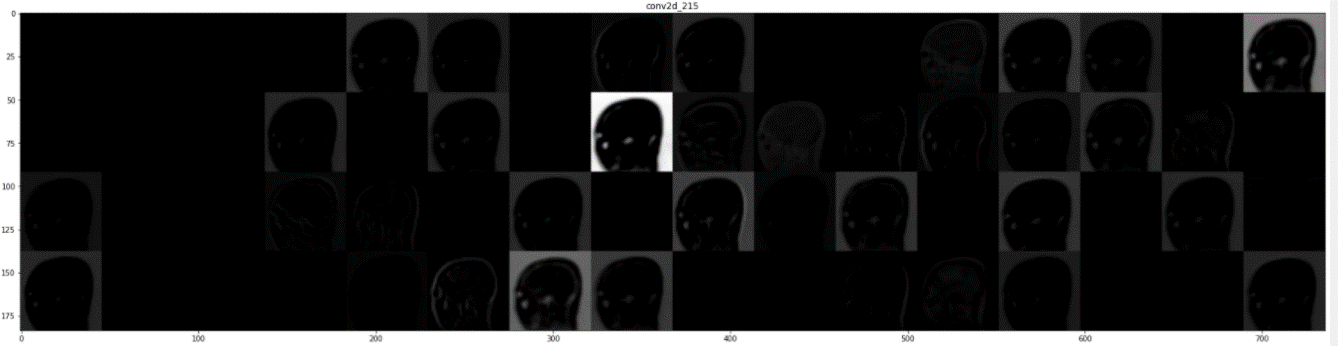
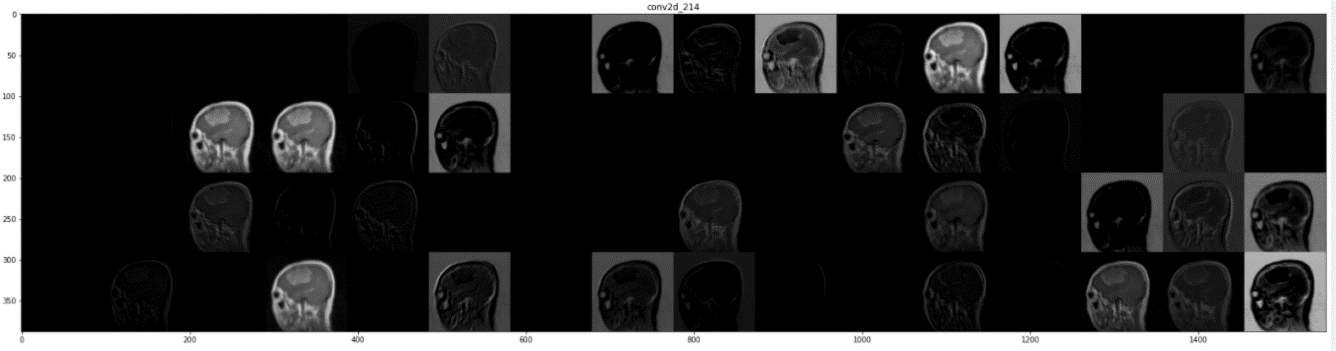
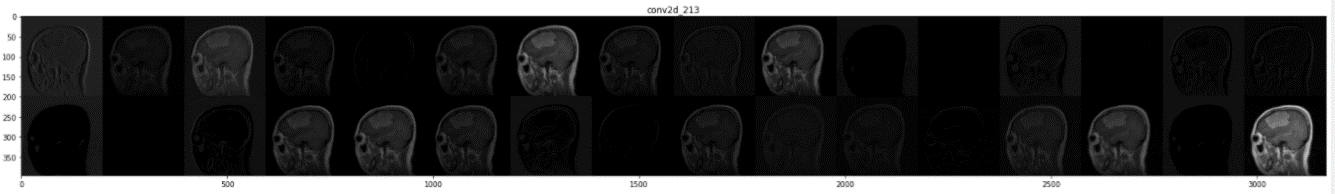
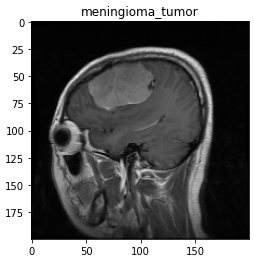
1. **Quantify what features/filters are more important to the classification of which tumor types?**

Glioma Tumor:



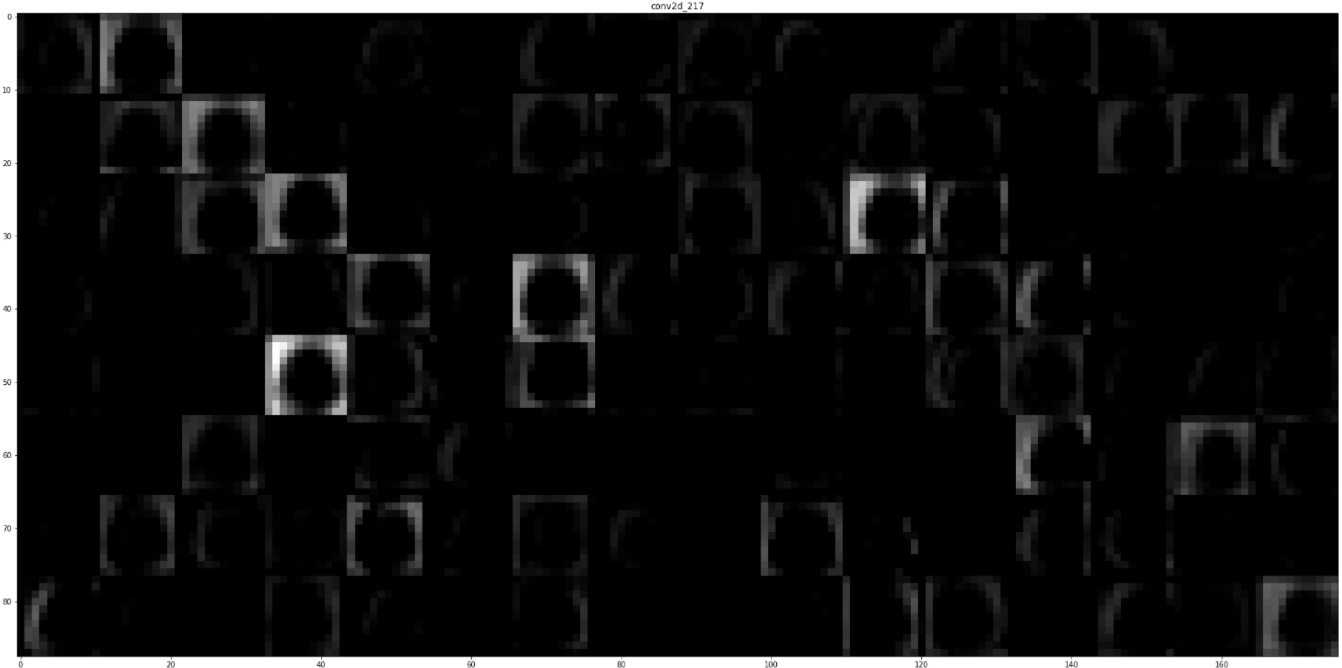
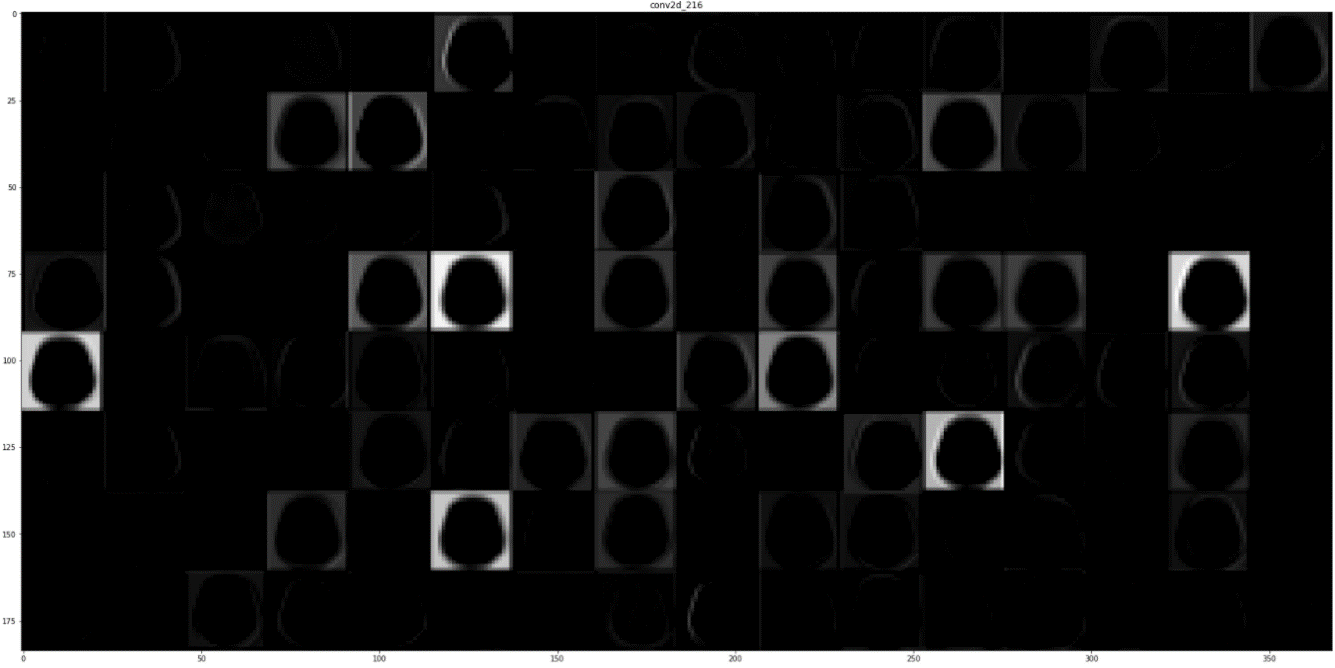
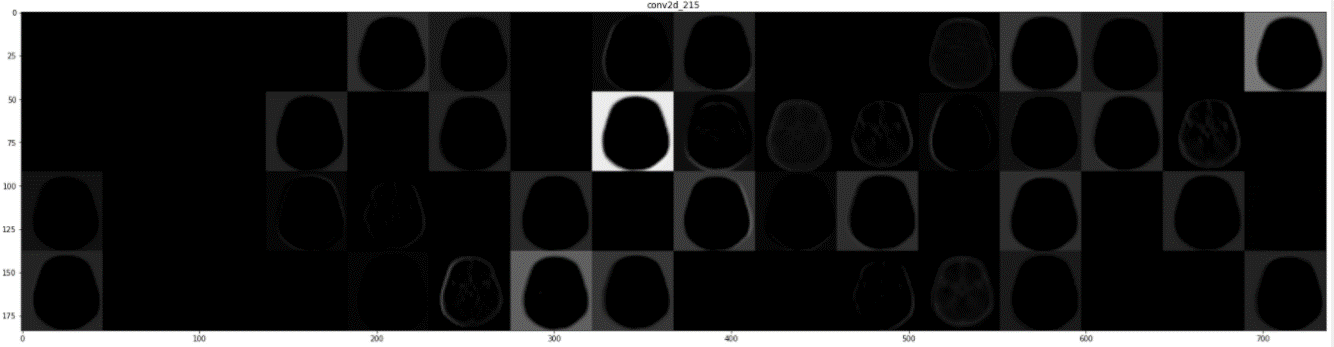
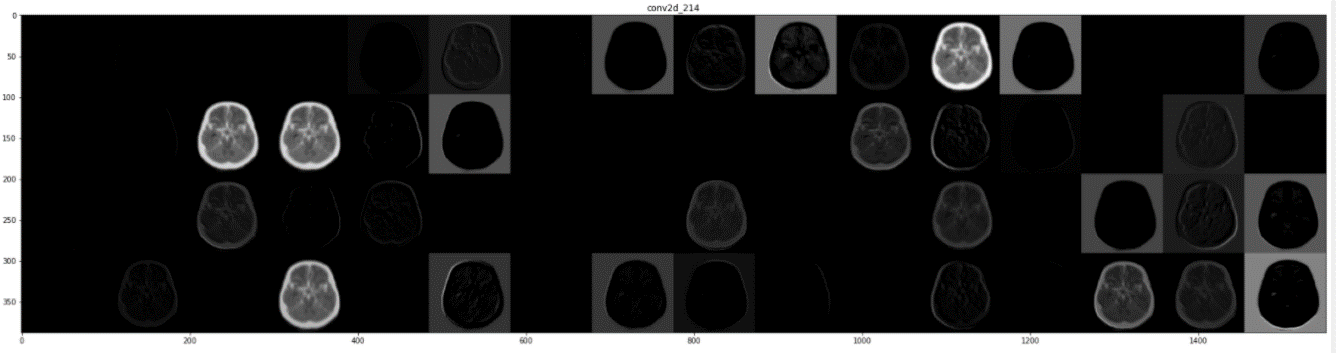
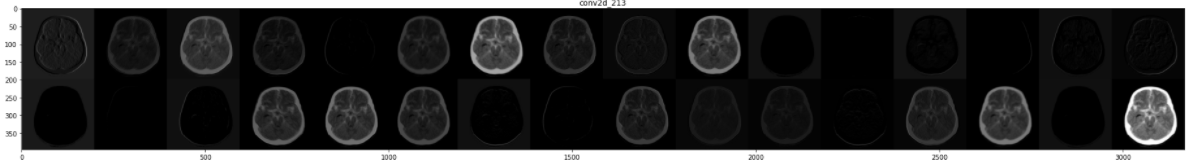
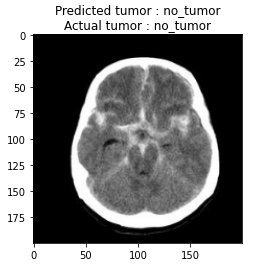
Presence of white dots at the left bottom part of the image.

Meningioma Tumor:



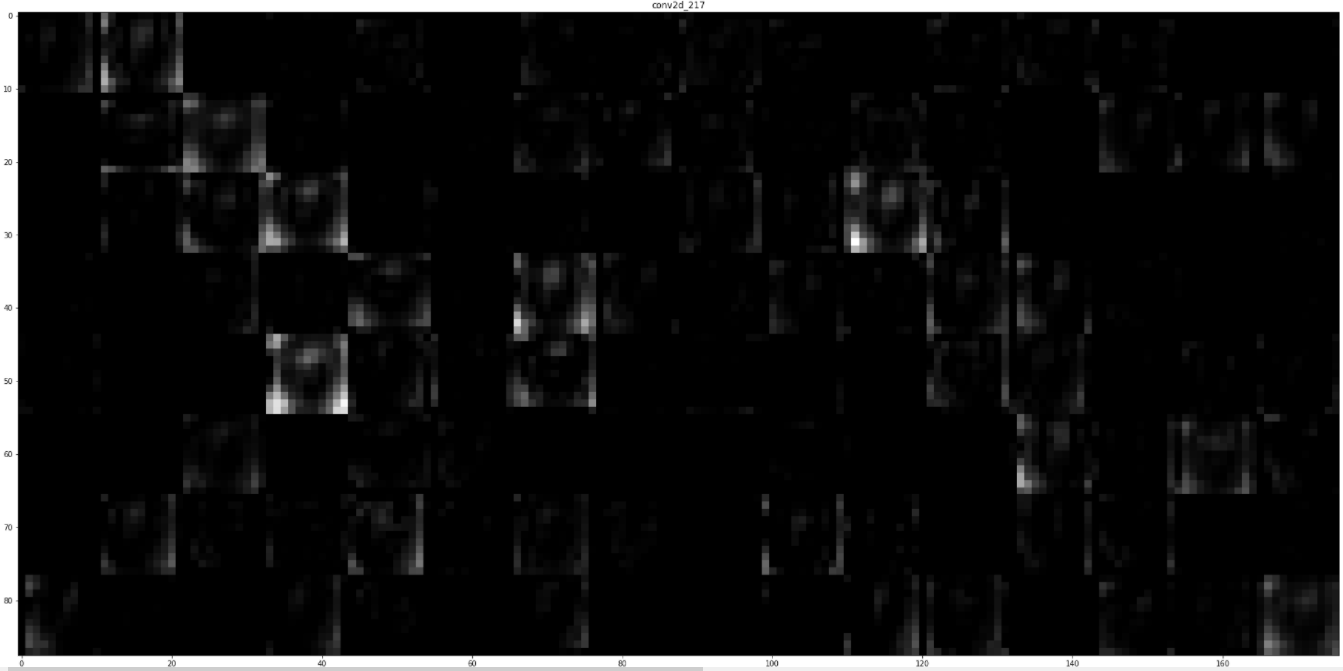
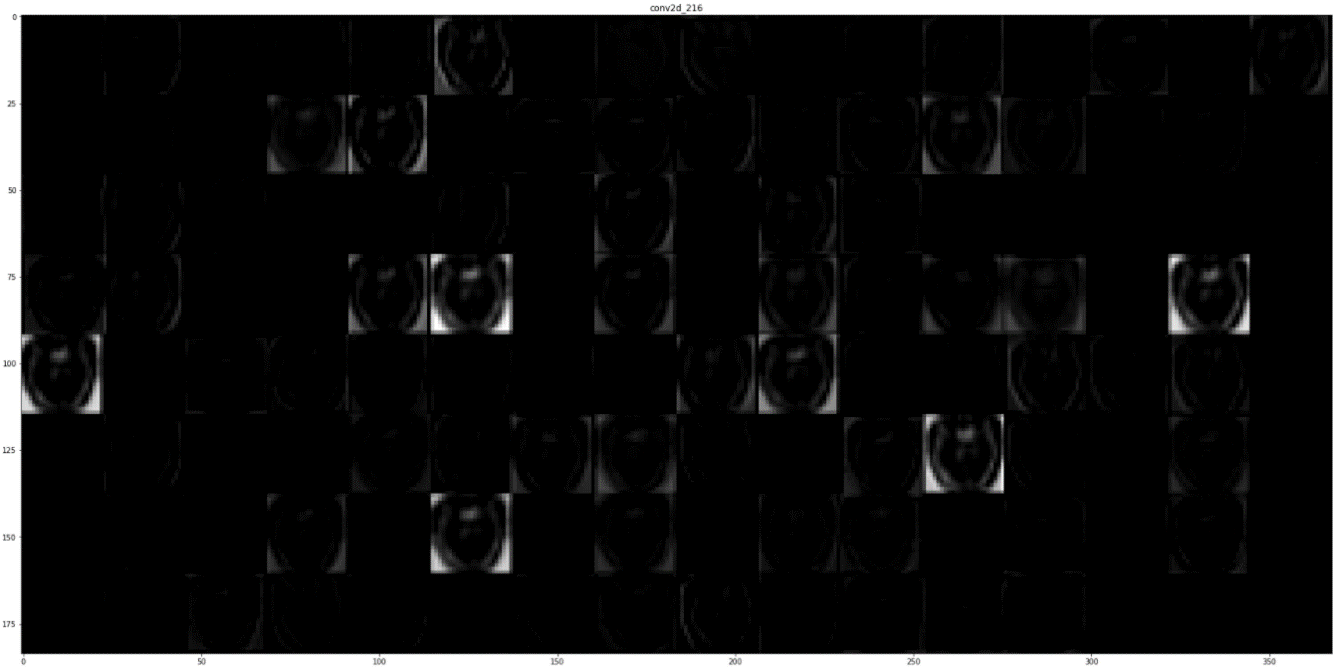
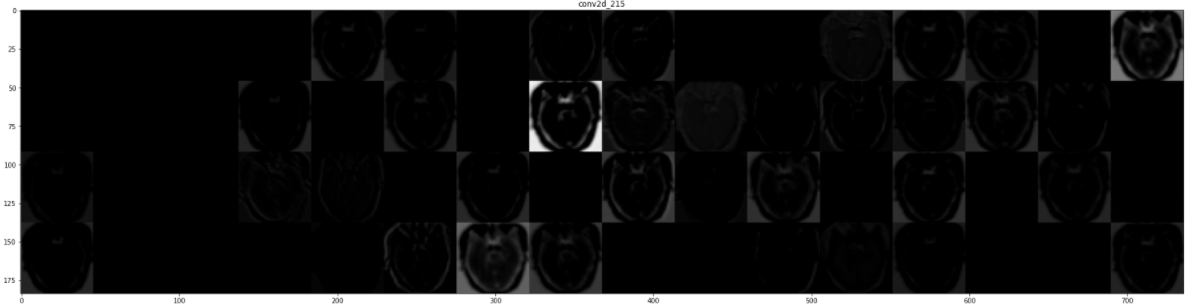
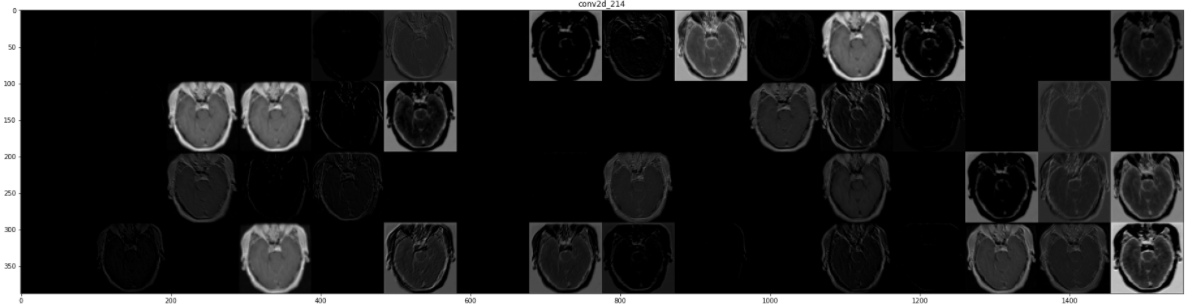
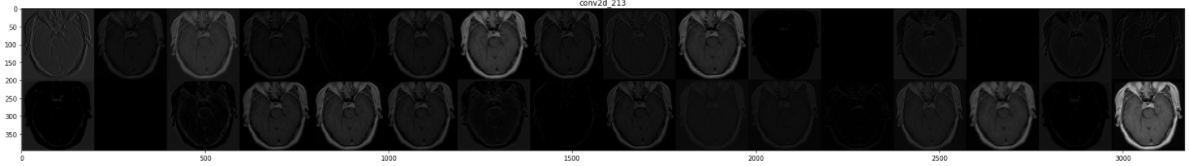
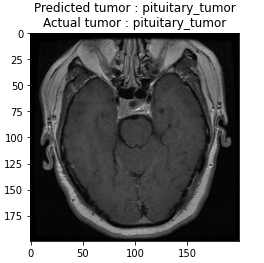
Presence of white dot like structure at the center of the image.

No Tumor:



No any dot-like structure in the image.

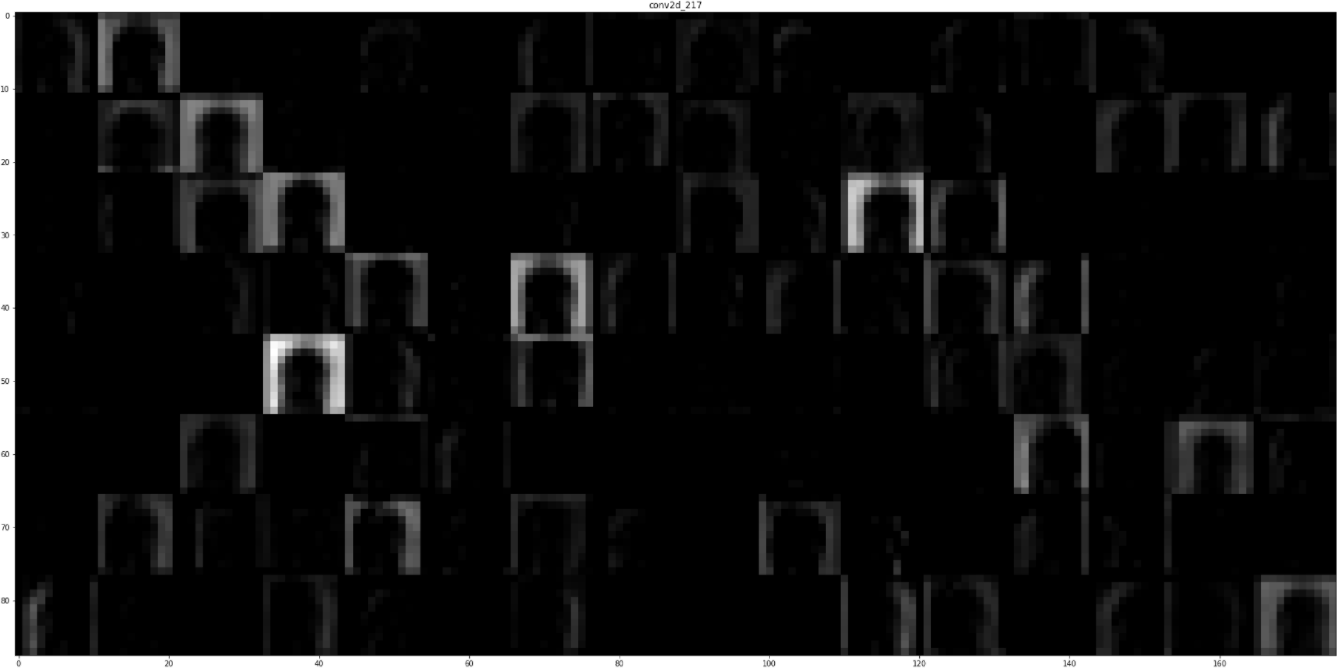
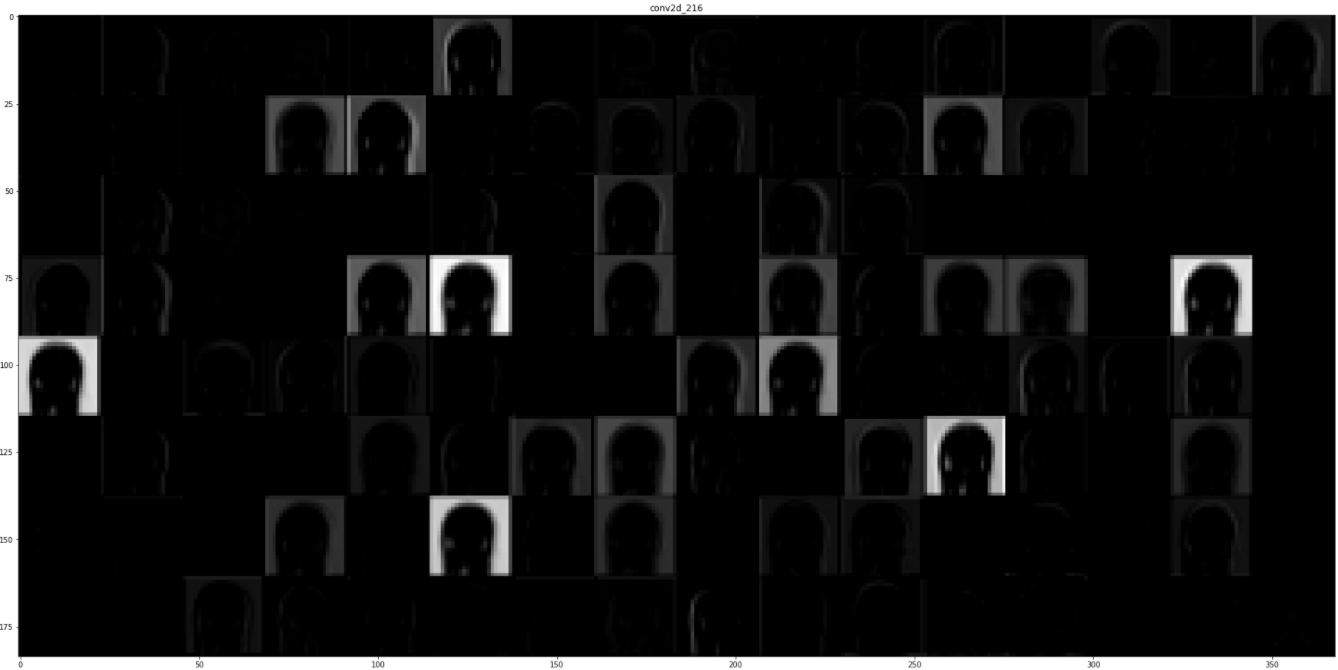
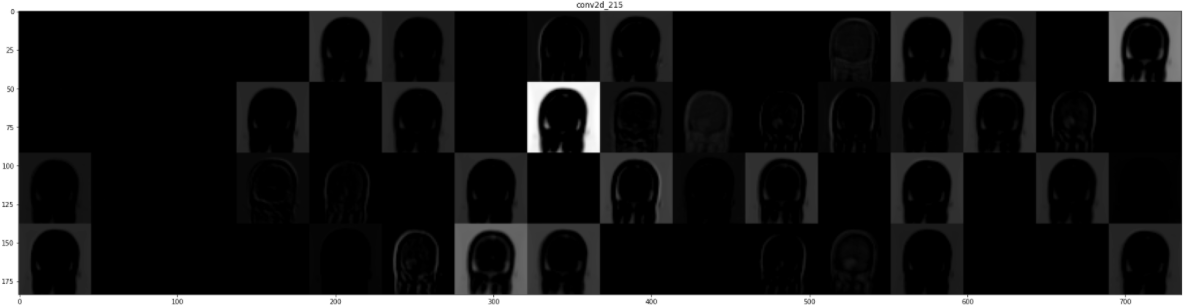
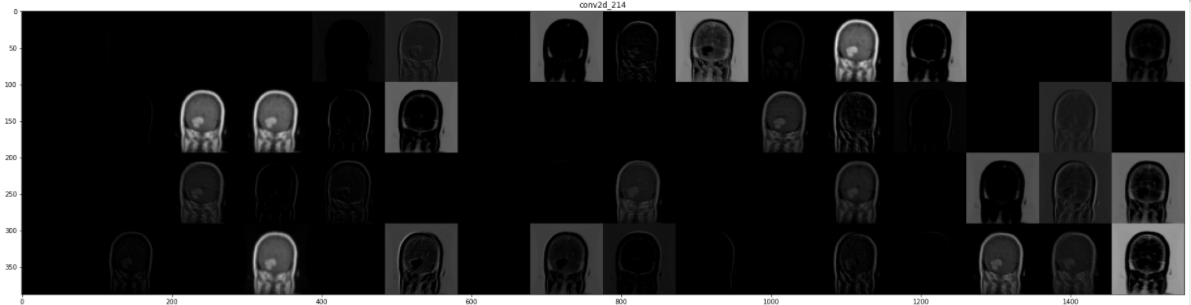
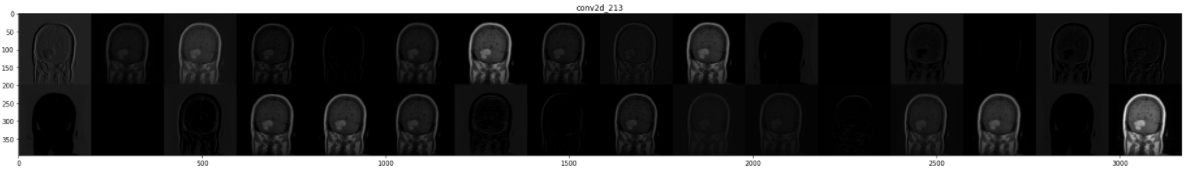
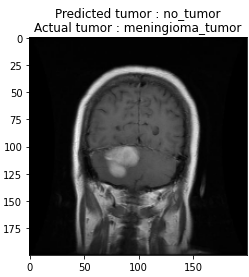
Pituitary Tumor:



Presence of white dots at top part of image.

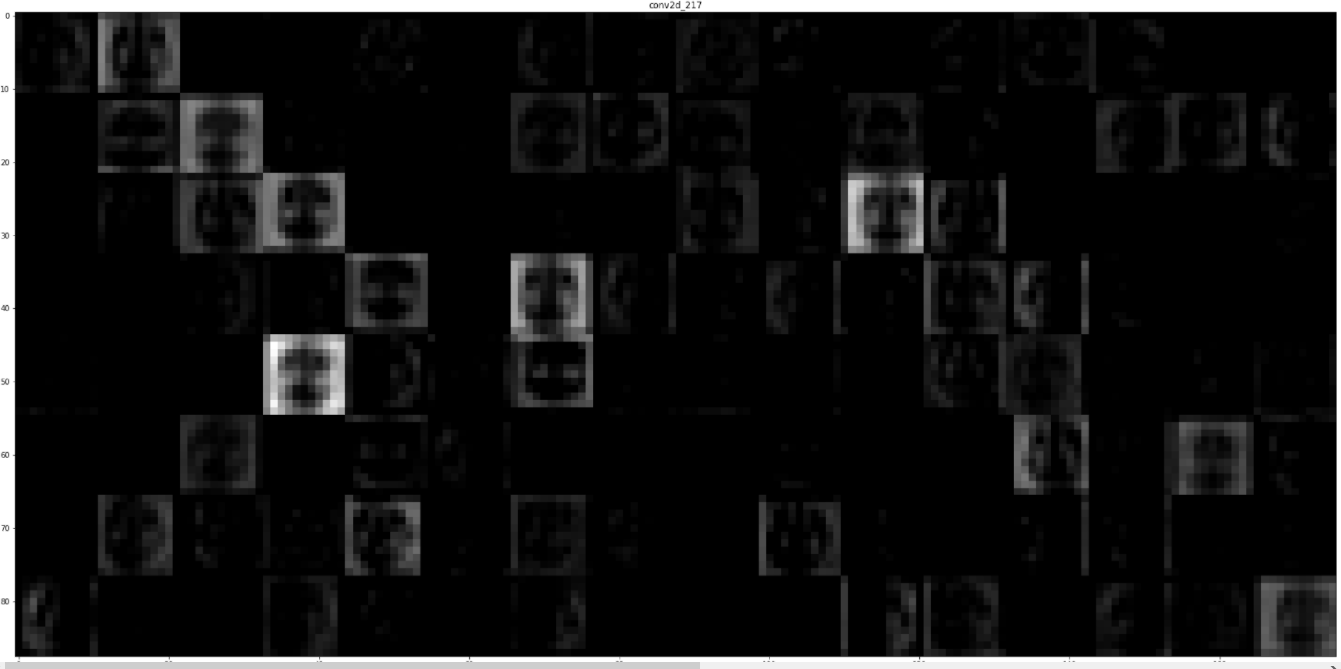
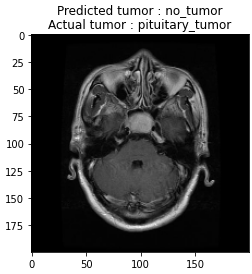
1. **Can you use these filters to improve the performance of your network?**

**Meningioma tumor to No tumor classification:**



In actual Meningioma classification important features was presence of dot-like structure at the center of image but, in this case, there are no any such structures at center which belong to No tumor class.

**Pituitary tumor to No tumor classification:**



In actual Pituitary tumor classification, there was presence of white dots at top but, there are no significant such structures at top and dots at side are connected to outer and it does not match with other classification leaving it to No tumor classification.