Norwegian University
of Life Sciences

Master’s Thesis 2020 30 ECTS
Falcuty of Science and Technology
Professor Cecilia Marie Futsæther

Visualization of deep learning in auto-delineation of cancer tumors.

Bao Ngoc Huynh
MSc Data Science

This page is intentionally left blank.

Acknowledgements

First of all, this project can never be completed without the guidance of my supervisor, Prof. Cecilia Marie Futsæther. She has been an enthusiastic supporter and has provided me with continuous and thoughtful feedback.

Moreover, I thank Yngve Mardal Moe, who inspired me with the initial ideas of the project and gave constructive recommendations throughout the development of the project.

Furthermore, I thank Ms. Aurora Grøndahl, Mr. Stefan Schrunner, Prof. Oliver Tomic, Prof. Kristian Liland, for their participation in meetings regarding my project with helpful feedback and guidance.

This project would be impossible without the availability of the dataset. Therefore, I thank oncologist Dr. Einar Dale and Prof. Eirik Malinen for making the dataset accessible.

I thank my sister, Bao Ngan Huynh, who took care of me during the stressful days of programming and writing.

Finally, I thank my parents, who have supported me throughout my life.

Bao Ngoc Huynh
Ås, May 30th 2020

Abstract

Purpose
The *deoxys* framework, developed by Huynh and is available at `https://github.com/huynhngoc/deoxys/`, has the final goal of creating a user-friendly software that helps radiologists with tumor delineation problems. Currently, users of this framework can create and run any deep learning experiments using this framework. To increase the transparency and interpretability of the deep learning model, there is a need for adding model visualization methods into the *deoxys* framework. Therefore, in this thesis, several model visualization methods were implemented and integrated into the *deoxys* framework. In addition, this thesis also demonstrated the benefits of model visualization for users of the *deoxys* framework, including radiologists and data scientists.

Methods
Model visualization methods such as activation maps, activation maximization, saliency maps, deconvnet, and guided backpropagation were implemented in this Master's thesis. The implementation of these visualization methods was assessed by comparing the results provided by the *deoxys* framework to previously published results.

The implemented visualization methods were applied to a deep learning model, which was trained on the head and neck cancer data of PET/CT scans for automatic tumor cancer delineation. The model visualization results were interpreted to demonstrate their benefits for model understanding.

Results
The implementation of the model visualization methods reproduced similar results with previous studies, thus passed the quality control assessment and ensured the reliability of the implemented visualization methods.

When interpreting the visualization results, the pretrained model was found to extract the tongues, bones, muscles and glands from the CT scans and the lymph nodes from the PET scans. In addition, the pretrained model had a high chance of marking a pixel as cancer tumors if that pixel belonged to the bright lymph nodes in the PET scan. Moreover, the weakness of the pretrained model such as the lack of data augmentation was found during the interpretation.

Conclusions
The model visualization methods were demonstrated to benefits both radiologists and data scientists. Radiologists can have internal insights of the deep learning model, while data scientists can find existing problems to improve the deep learning model performance.

Despite the existing limitations, the developed *deoxys* framework has the potential for improvement. This includes enhancement of the implemented visualization methods, and the addition of other model visualization methods. Ideally, an interactive user interface should be developed to satisfy the user-friendly goal of the *deoxys* framework.

Contents

1 Introduction **1**
1.1 Motivation 1
1.1.1 Deep learning and automatic delineation of cancers tumors . 1
1.1.2 Visualization of deep learning model 2
1.2 Aims of this Master's thesis 3

2 Visualization of Deep Learning **5**
2.1 Deep Learning 6
2.1.1 Neural Networks 6
2.1.2 Activation function 9
2.1.3 Loss function 11
2.1.4 Forward and backward propagation 11
2.2 Convolutional Neural Network 14
2.2.1 Images and Tensors 14
2.2.2 Filter operations 14
2.2.3 The convolutional layer 15
2.3 Visualization 18
2.3.1 Activation Map 18
2.3.2 Activation Maximization 21
2.3.3 Saliency Map, Deconvnet and Guided Backpropagation . . . 24

3 Code **29**
3.1 Deoxys Framework 29
3.1.1 Introduction and usage 29
3.1.2 Structures 30
3.2 Updates 34
3.2.1 New APIs for model visualization 34
3.2.2 Quality Control 35
3.2.3 Others extension 36

4 Experimental setup **39**

4.1 The dataset and the pretrained model 39
  4.1.1 Head and neck cancer dataset 40
  4.1.2 The Model 40
4.2 Visualization 43
  4.2.1 Activation maps 45
  4.2.2 Activation Maximization 45
  4.2.3 Gradient-based visualization 45

5 Results 47
5.1 Implemented updates of *deoxys* 47
  5.1.1 Visualization 47
  5.1.2 Database Integration 51
5.2 Visualization Results 53
  5.2.1 Patterns extracted from the Unet model 53
  5.2.2 Pixel contributions to the delineation 69

6 Discussion 83
6.1 The Unet model 83
  6.1.1 Unet Model performance 84
  6.1.2 Model visualization results 86
6.2 Benefits of model visualization 89
  6.2.1 For radiologists 89
  6.2.2 For data scientists 90
6.3 Application to other cancers 90
6.4 Difficulty in the implementation process 91
6.5 Potential for improving the deoxys framework 92
  6.5.1 Improvement of existing visualization methods 92
  6.5.2 Addition of visualization methods 93
  6.5.3 User interaction 94

7 Conclusion 97

Appendices

A DAT390 Data Science Seminar report 107

B Models trained on ImageNet dataset 119

C Database test report 123

D The detailed U-net model structure 127

E Full activation maps **131**
E.1 Activation maps patient 91, slice 86 131
E.2 Activation maps patient 148, slice 11 148
E.3 Activation maps patient 209, slice 14 165
E.4 Activation maps patient 217, slice 20 182
E.5 Activation maps patient 233, slice 48 199
E.6 Activation maps patient 249, slice 55 216

F Gradient-based visualization **233**

List of Figures

2.1 Illustration of a three-layer neural network 7
2.2 Illustration of data transformation of a node in the neural network . 7
2.3 Illustration of a general neural network 8
2.4 Illustration of forward and backward propagation in the neural network 13
2.5 Illustration of the convolution operation on a 2D image 16
2.6 Illustration of the process of generating activation maps 19
2.7 Illustration of an example of activation maps from the VGG16 model 20
2.8 Illustration of the process of activation maximization 22
2.9 Example of activation maximization of VGG16 model. 23
2.10 Illustration of the process of generating saliency maps, deconvnet and guided backpropagation 25
2.11 Example results of saliency map, deconvnet and guided backpropagation 27

3.1 Use case diagram of the *deoxys* framework 32
3.2 Structure of the *deoxys* framework 33
3.3 Relationship between objects in the database 37

4.1 Illustration of the Unet architecture 42

5.1 Illustration of VGG16 activation maximization results 49
5.2 Quality control results on saliency maps, deconvnet and guided backpropagation 50
5.3 Screenshots from the example terminal application integrated with database 52
5.4 Activation Maps of image slice 86 from patient 91. 54
5.5 Activation Maps of image slice 11 from patient 148. 56
5.6 Activation Maps of image slice 14 from patient 209. 58
5.7 Activation Maps of image slice 20 from patient 217. 60
5.8 Activation Maps of image slice 48 from patient 233. 62

5.9 Activation Maps of image slice 55 from patient 249. 64
5.10 Features extracted from different input images using the same filter 66
5.11 Activation maximization results at some filters in conv2d_3 layer of the Unet model 68
5.12 Activation maximization results at some filters in conv2d_17 layer of the Unet model 69
5.13 Saliency maps using *positive prediction* loss function of images with high Dice score 72
5.14 Deconvnet-based visualization results using *positive prediction* loss function of images with high Dice score 73
5.15 Guided backpropagation-based visualization results using *positive prediction* loss function of images with high Dice score 74
5.16 Saliency maps using *positive prediction* loss function of images with 0.5 Dice scores 75
5.17 Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores 76
5.18 Saliency maps using *true positive* loss function of images with 0.5 Dice scores 77
5.19 Guided backpropagation-based visualization results using *true positive* loss function of images with 0.5 Dice scores 78
5.20 Saliency maps using *positive prediction* loss function of images with 0.0 Dice scores 79
5.21 Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores 80
5.22 Saliency maps using *true positive* loss function of images with 0.0 Dice scores 81
5.23 Guided backpropagation-based visualization results using *true positive* loss function of images with 0.0 Dice scores 82

E.1 Full activation maps of image slice 86 from patient 91 132
E.2 Full activation maps of image slice 11 from patient 148 149
E.3 Full activation maps of image slice 14 from patient 209 166
E.4 Full activation maps of image slice 20 from patient 217 183
E.5 Full activation maps of image slice 48 from patient 233 200
E.6 Full activation maps of image slice 55 from patient 249 217

F.1 Saliency maps using *positive prediction* loss function of images with high Dice scores 234
F.2 Deconvnet-based visualization results using *positive prediction* loss function of images with high Dice scores 235

F.3 Guided backpropagation-based visualization results using *positive prediction* loss function of images with high Dice scores 236
F.4 Saliency maps using *true positive* loss function of images with high Dice scores 237
F.5 Deconvnet-based visualization results using *true positive* loss function of images with high Dice scores 238
F.6 Guided backpropagation-based visualization results using *true positive* loss function of images with high Dice scores 239
F.7 Saliency maps using *positive prediction* loss function of images with 0.5 Dice scores 240
F.8 Deconvnet-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores 241
F.9 Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores 242
F.10 Saliency maps using *true positive* loss function of images with 0.5 Dice scores 243
F.11 Deconvnet-based visualization results using *true positive* loss function of images with 0.5 Dice scores 244
F.12 Guided backpropagation-based visualization results using *true positive* loss function of images with 0.5 Dice scores 245
F.13 Saliency maps using *positive prediction* loss function of images with 0.0 Dice scores 246
F.14 Deconvnet-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores 247
F.15 Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores 248
F.16 Saliency maps using *true positive* loss function of images with 0.0 Dice scores 249
F.17 Deconvnet-based visualization results using *true positive* loss function of images with 0.0 Dice scores 250
F.18 Guided backpropagation-based visualization results using *true positive* loss function of images with 0.0 Dice scores 251

List of Tables

3.1 Materials for test cases. 38
3.2 Example of a test case in database integration. 38

4.1 The number of patients in each of the datasets. See [11] for details . 40
4.2 Information of images with high Dice score for interpretation. . . . 43
4.3 Information of images with Dice score of 0.5 (intermediate performance) for interpretation. 43
4.4 Information of images with Dice score of 0.0 for interpretation, indicating that the model delineations for these slices did not overlap with the ground truth. 44
4.5 Layers in the Unet architecture (Figure 4.1) used for model interpretation. 44

6.1 Comparison of the results of tumor segmentation between the *deoxys* framework and other studies using the Dice score (DS). The first group contains the results of the head and neck cancer tumor segmentation. The second group contains the results of specific types of head and neck cancer tumor segmentation. The final group contains the results of brain tumor segmentation. 85

B.1 The detailed structure of the VGG16 model [37]. 120
B.2 The detailed structure of the model used by Springenberg *et al.* in [20]. 121

D.1 Overview of the architecture used to train on the head and neck cancer dataset. The downsampling path is given in the table. 128
D.2 Overview of the architecture used to train on the head and neck cancer dataset. The bottleneck is given in the table. 129
D.3 Overview of the architecture used to train on the head and neck cancer dataset. The upsampling path is given in the table. 129
D.4 Overview of the architecture used to train on the head and neck cancer dataset. Continued from Table D.3 130

Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
Conv	Convolution
CNN	Convolutional Neural Network
CT	(X-Ray) Computerised Tomography
DBMS	Database Management System
HDF(5)	Hierarchical Data Format (5)
HU	Hounsfield Unit
JSON	JavaScript Object Notation (a standard data serialization format)
PET	Positron Emmision Tomography
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
SUV	Standardized Uptake Value

Mathematical notation

Mathematical symbol	Meaning
Σ	The linear combination of the input nodes of a layer
$\mathcal{W}_i$	The weights matrix of the i^{th} layer
$\phi(x)$	An activation function
$\phi_{ReLu}(x) = max(0, x)$	The ReLU activation function
$\phi_{sigmoid}(x) = \frac{1}{1+exp(-x)}$	The sigmoid activation function
$\phi_{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	The softmax activation function
$\phi_{LeakyReLu}(x) = \begin{cases} \alpha \cdot x \text{ if } x < 0, \\ x \text{ otherwise} \end{cases}$	The LeakyReLu activation function
C^i	The output of the convolution operation in the convolutional layer
F^i_{In}	Input of the i^{th} layer
F^i_{out}	Output of the i^{th} layer
$R^{i+1} = \frac{\partial L^{i+1}_{out}}{\partial L^{i+1}_{in}}$	The gradients of the model output with respect to the i^{th} layer
$R^i_{saliency} = (C^i > 0) \cdot R^{i+1}$	Gradients used in the saliency maps at the i^{th} layer
$R^i_{deconvnet} = (R^{i+1} > 0) \cdot R^{i+1}$	Gradients used in the deconvnet method at the i^{th} layer
$R^i_{guided\_backpropagation} = (C^i > 0) \cdot (R^{i+1} > 0) \cdot R^{i+1}$	Gradients used in the guided backpropagation method at the i^{th} layer

Chapter 1

Introduction

1.1 Motivation

1.1.1 Deep learning and automatic delineation of cancers tumors

Radiotherapy[1] is the most common treatment for cancer, a deadly disease which caused the death of over 9 million people in 2018 [1]. In radiotherapy, cancer cells are killed by high-energy radiation, such as X-rays, or gamma-rays. However, in the irradiation process, not only are cancer cells killed, but the neighboring healthy tissues can also be affected. For that reason, accuracy in cancer tumor delineation is essential in this kind of treatment [2][3]. Furthermore, because of the inter-observer variability, when different radiologists delineate the same cases, the variation of gravity centers of cancer tumors can be larger than patient positioning and organ motion [4][5]. Therefore, having more than one radiologist for each case can help to increase the accuracy in delineation. However, this method is almost impossible due to the long waiting time to delineate one case [6] and the lack of human resources [7].

With the innovation of technology in recent years, deep learning has been applied to cancer detection, classification and tumor delineation and has obtained high accuracy[2] [8]. Many deep learning models for tumor delineation have been

---

[1] In some other publications, this term is referred as Radiation therapy.

[2] The accuracy is calculated by using human-based results as the ground truth.

proposed for head and neck cancer [9][10][11], rectal cancer [12], lung cancer [13] and anal cancer [14]. Thus, the delineation results from deep learning can act as an independent observer to help radiologists to delineate cancer tumors faster, consistently and with higher accuracy.

1.1.2 Visualization of deep learning model

The main stakeholders of the deep learning model for auto-delineation of cancer tumors are radiologists, who use the delineation results from deep learning as references, and data scientists, who propose and build the deep learning model. When radiologists use the delineation results as references, it would be more helpful if they can see how the machine "sees" and "interprets" the medical images. Also, not all deep learning models are perfect, and data scientists always want to improve the proposed models as much as possible. This lead to the need for visualizing the deep learning model[3].

Model visualization can be used to see how the images transform in the deep learning model. From there, the radiologist can see which parts of the images are extracted during each step in the model, as well as how the model makes the decision. In addition, parts of the images that have large effects on the results of the deep learning model can be found using other visualization methods. This benefits both radiologists, who can see an interpretation of the model, and data scientists, who can find the weakness of the proposed model. Therefore, visualization of the deep learning model would be helpful for both radiologists and data scientists, as it eases interpretation and makes the approach more useful for medical specialists.

Nowadays, many methods of visualization have been proposed. For example, the method named activation maximization [15] visualizes the features of the images that the models extract. This method can be improved by adding regularization [16] and priors [17]. Saliency maps [18], Deconvnet [19] and Guided Backpropagation [20] are visualization methods to find parts of the images that are important for the prediction of the model. Based on these methods, Class Activation Maps (CAM) [21] and Grad-CAM [22] are proposed with similar goals.

---

[3] The term visualization of deep learning model used in this thesis refers to visualization methods that explain the deep learning models, mainly focusing on the data flow in the model, as well as the aspects that influence the results of the models. This does not include visualization of the structure of the model, or the performance of the model.

1.2 Aims of this Master's thesis

In preparation for this Master's thesis, the author developed a Keras-based [23] deep learning framework with the ability to create and train different deep learning models, including models for automatic delineation of cancer tumors (see Appendix A on page 107). One of the goals of this Master's thesis is to update this framework with model visualization features, including the options to choose from several methods such as activation maps [24], activation maximization [15], saliency maps [18], deconvnet [19] and guided backpropagation [20]. The other goal of this thesis is to demonstrate the benefit of model visualization, by interpreting the visualization of a deep learning model which was trained on head and neck cancer data.

In this thesis, firstly, Chapter 2 provides a fundamental context of deep learning. This includes deep learning on image data, as well as explanations of several model visualization methods. Then, Chapter 3 introduces an overview of the developed deep learning framework. This chapter also provides the proposed updates relating to model visualization, along with other extensions of the framework. The quality controls to assess the updates are also explained in this chapter. Thereafter, Chapter 4 describes the trained model to be interpreted, together with the head and neck cancer dataset. This chapter also describes the visualization methods that are used on the pre-trained model. Chapter 5 shows the results of the proposed updates of the framework. In addition, the results of the visualization process can also be found in this chapter. The interpretation of the visualization results, the benefits of model visualization, and the potential of the developed framework, are discussed in Chapter 6. Finally, Chapter 7 states the conclusions of this thesis.

Chapter 2

Visualization of Deep Learning

The term *Machine Learning*, which is a sub-field of *Artificial Intelligence*, refers to the actions of finding rules from existing data in order to make predictions [25] [24]. These rules, called the *machine learning model*, or in short, *model*, is the function mapping the input data with the output answers. Thus, a machine learning system makes predictions by transforming the input data through the *machine learning model*. Also, a machine learning system learns from existing data by using these data as feedback signals to modify the *model* so that the prediction outputs are as close to the actual outputs as possible. In other words, the goal of the learning algorithm in a machine learning system is to achieve a model with the best performance.

The term *Deep Learning* refers to an approach in *Machine Learning*, in which the machine learning model is in the form of a *Neural Network* with a number of layers [25] [24]. The number of layers in the *Neural Network* determines the depth of the *Neural Network*, which explains the "deep" part in the term *Deep Learning* [24].

This chapter provides the fundamental context of Deep Learning. This includes the structure of a general network and its learning algorithm, as well as the components of the convolutional neural network (CNN) for using deep learning on image data. In addition, this chapter also explains different visualization methods based on the components of the neural network.

2.1 Deep Learning

2.1.1 Neural Networks

Figure 2.1 illustrates the data flow in a simple neural network with three layers: the *Input Layer*, the *Output Layer*, and one *Hidden Layer* in between. This neural network predicts two types of output, y_1 and y_2, from the initial input of two features[1] x_1 and x_2. The circles in each layer represent the nodes in that layer, while the arrows represent the data flow. Each node performs a data transformation from the outputs of the previous layers. Data transformation starts by first going through the *Input Layer*. This layer does not make any changes to the data. Thus, the number of nodes in this layer depends on the number of features of the initial input. After that, the outputs of the *Input Layer*, with the addition of the *bias* node, act as inputs for the next layer, the *Hidden Layer*. Finally, the Hidden Layer's outputs, together with a *bias* node, act as inputs to the *Output Layer* for predictions.

To understand each node's data transformation, we can take a closer look at the red part of Figure 2.1, which is illustrated in Figure 2.2. First, the linear combination of the previous layer's outputs, together with the *bias* node, is calculated. This can be denoted as:

$$\begin{aligned}\Sigma &= w_0 \cdot 1 + w_1 l_1 + w_2 l_2 \\ &= w_0 l_0 + w_1 l_1 + w_2 l_2 \qquad \text{(denote bias node's value as } l_0\text{)} \\ &= \mathbf{w}^T \mathbf{L}\end{aligned}$$

$$\text{where the weights } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \text{ and the layer's input } \mathbf{L} = \begin{bmatrix} l_0 \\ l_1 \\ l_2 \end{bmatrix}$$

After that, an *activation function*[2], denoted as $a = \phi(x)$, is applied to the resulting linear combination Σ, called the *weighted sum*, to calculate the output of the node. Because of that, the nodes in each layer in the neural network are called *activation units*. Since $\mathbf{w}$, called the *weights*, directly affect the transformation of data in the neural network, the goal of training the neural network is to find the weights that give the model the best performance.

---

[1] This term is sometimes referred as *variables* or *columns*.

[2] *Activation function* will be introduced in Section 2.1.2 on page 9.

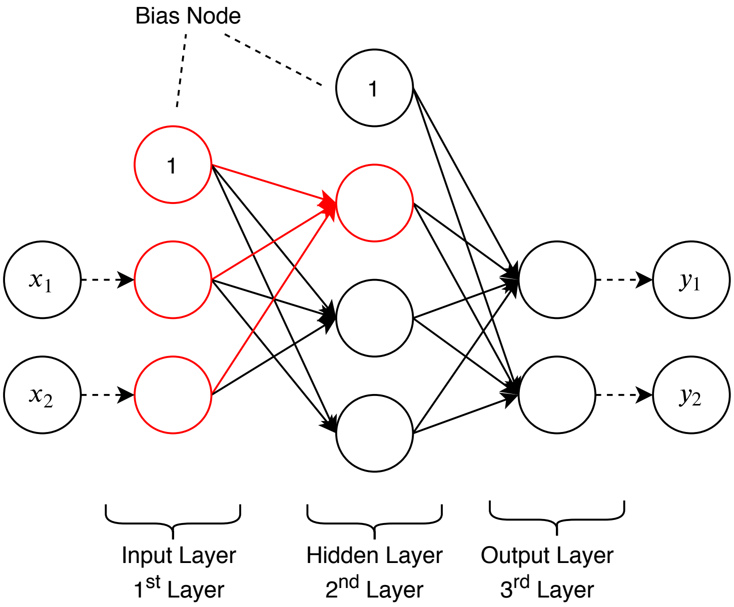


Figure 2.1: Illustration of a three-layer neural network. This neural network predicts two types of output, y_1 and y_2, from the initial input of two features x_1 and x_2. The circles in each layer represent the nodes in that layer, while the arrows represent the data flow. Each node performs a data transformation from nodes of the previous layers. The detailed information in red parts is illustrated in Figure 2.2.

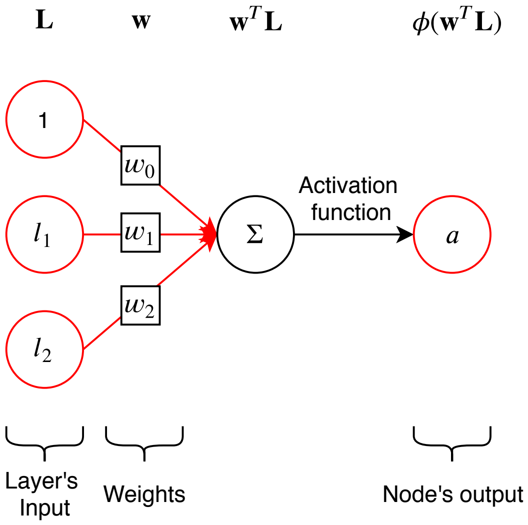


Figure 2.2: Illustration of data transformation of a node in the neural network. The red part is also associated with the red part in Figure 2.1. First, the linear combination of the inputs is calculated. After that, an activation function is applied to calculate the output of the node.

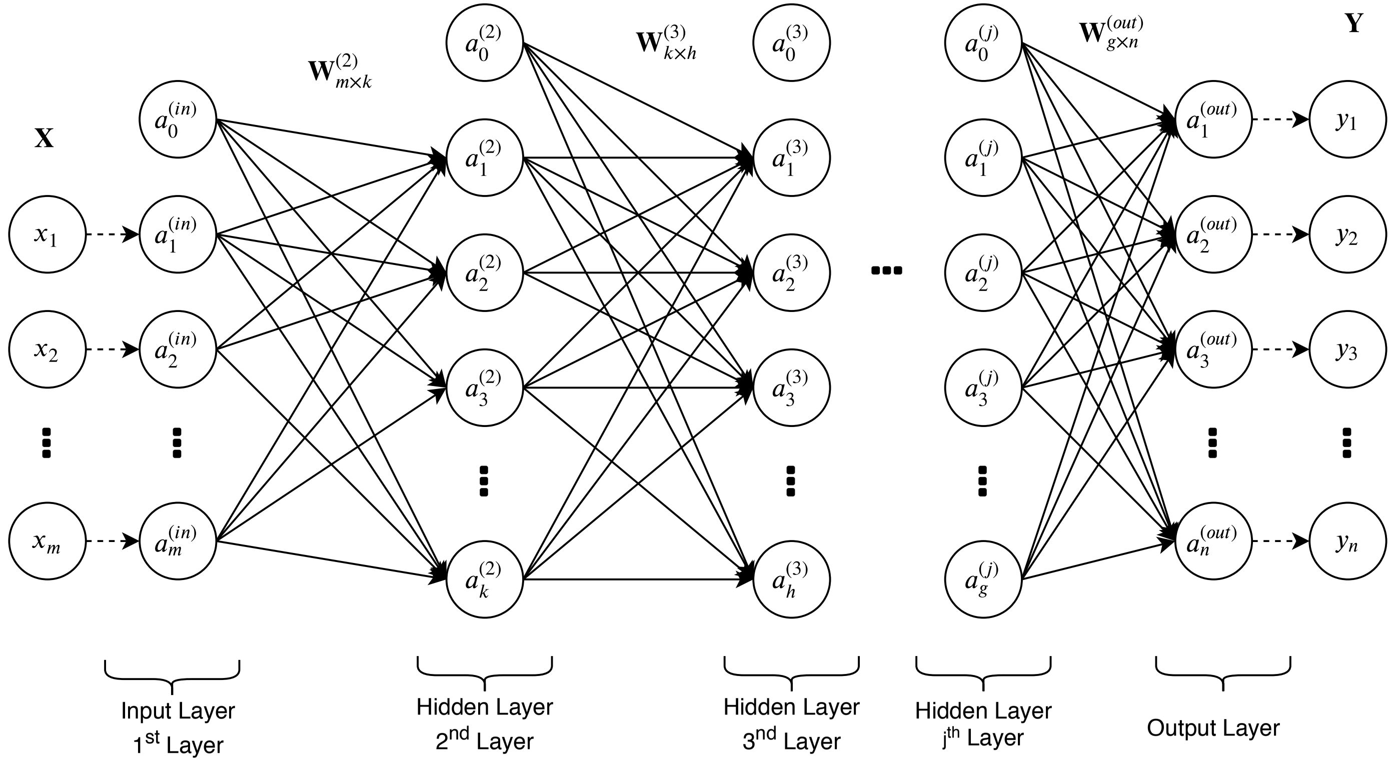


Figure 2.3: Illustration of a general neural network. It contains an Input Layer, an Output Layer and a number of Hidden Layers. $a_i^{(j)}$ denotes the i^{th} activation unit of the j^{th} layer. $\mathbf{W}_{s\times t}^{(j)}$ denotes the weights matrix between the $(j-1)^{th}$ layer with s units and the j^{th} layer with t units.

Generally, a neural network contains an *Input Layer*, an *Output Layer*, and a number of *Hidden Layers* in between (Figure 2.3). In the artificial neural network, data is transformed as it goes through each layer, which contains a number of nodes. Each node is calculated by first obtaining the linear combination of the outputs of the nodes from the previous layer, then applying the activation function on the value obtained. These nodes are called *activation units*. Thus, the data flow is similar to a network in which the outputs of all nodes from the previous layer are the inputs of each node in the next layer. In short, in a neural network, the output of the previous layer is the input of the next layer, and the output of the final layer is the prediction of the input data. The final goal of the learning algorithm in the neural network is to find all weight matrices so that the final outputs are as close to the expected results as possible.

2.1.2 Activation function

The activation function is a continuous function applying to any layer in the neural network [24][26][27]. This function can be linear, in the form of $\phi(x) = cx$, or non-linear [27]. However, a layer with a linear activation function (layer L^a) is the same as a layer without activation functions (layer L^b). This is because both resulting outputs of these two layers are still linear combinations of the inputs, and the weights $\mathbf{w^a}$ of layer L^a is c times smaller than the weights $\mathbf{w^b}$ of layer L^b. Furthermore, when a neural network only contains linear activation functions, the "deep" part of the neural network is meaningless since the neural network's final outputs are just the linear combinations of the initial inputs regardless of the number of hidden layers in the neural network. Thus, the non-linear activation function plays an important role in the neural network. Thanks to the non-linearity properties, this kind of activation function helps to increase the important information and suppress the noise from the layer's inputs [27][28]. Besides, in classification problems, where the final outputs must be in a limited range, applying a non-linear activation function to the last layer of the neural network can solve these kinds of problems [25][26].

Since the types of activation functions used in the neural network have an impact on the neural network's performance and outputs [26][27], different problems require different kinds of activation functions. In this part, we will go through some frequently used activation functions in the neural network and when they are used.

ReLu

$$\phi_{ReLu}(x) = max(0, x)$$

The **re**ctified **l**inear **u**nit (ReLu) function [29] silences all negative values in the outputs of the layers. It has been a popular activation function in neural networks due to its dominant performance over the softmax [30] and the tanh [31] activation function [28]. The ReLu function is usually applied to the neural network's hidden layers in most classification and regression problems [26][28].

Sigmoid

$$\phi_{sigmoid}(x) = \frac{e^x}{e^x + 1}$$

The outputs of the sigmoid function [30] are always between 0 and 1. Thus, the sigmoid function is usually applied to the last layer of the neural network for binary classification problems (with only positive or negative values).

Softmax

$$\phi_{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The outputs of the softmax function [30] are the set of probabilities, all of which sum up to 100%. Hence, it is usually applied to the last layer of the neural network for multi-label classification problems. In these problems, the number of nodes in the output layer is the same as the number of labels. Each node in the layer output associates with the probability that the input item belongs to one label. Thus, the predictions of these models are based on the node with the highest probability in the output layer. For example, a neural network predicting if an input is a "cat", "dog" or "cow", with the associated outputs y_1, y_2 and y_3. If the outputs of the neural network are $y_1 = 0.1$, $y_2 = 0.75$ and $y_3 = 0.15$, the input item belongs to the "dog" label.

2.1.3 Loss function

As stated at the beginning of this chapter, the machine learning model is modified based on existing data as feedback. The *loss function* acts as the feedback signal by calculating the loss score, which can be understood as the distance between the prediction outputs and the actual outputs [24][31][32]. When training the model, the weights of the neural network are adjusted to minimize the loss score.

There are several types of loss functions. The Adaline model, the early form of the deep neural network but without the hidden layer, has the squared error function as the loss function [25][24]. Thus, it is the most well-known loss function used in deep neural networks for regression problems [25][31]. Besides the squared error loss function, there are other loss functions such as maximum likelihood for regression problems, and hinge and cross-entropy for classification problems [31][33]. Similar to the activation function, different problems also require different types of loss function as the choice of the loss function has significant impacts on the performance of the neural network [33].

2.1.4 Forward and backward propagation

The process of training a neural network contains forward and backward propagation (Figure 2.4). When training the model, the train data are transformed as they go through each layer. This process is called forward propagation [24]. After data transformation finishes in the last layer, the loss score is calculated based on the final outputs. Then, the weights of the neural network are modified based on the loss score calculated. The process of adjusting the weights based on the loss score is called backward propagation, or in short, backpropagation [24]. These two processes are repeated several times until the loss score is minimized.

The relationship between the weights matrix and the loss score can be denoted as:

$$\begin{aligned} outputs &= f(weights, inputs) \\ loss &= g(outputs, targets) = g(f(weights, inputs), targets) \end{aligned}$$

where f is the model while g is the loss function.

Because (1) the weights act as coefficient values in the loss function, and (2) the loss function is differentiable, we can decrease the loss score by moving the coefficient values, or the weights, in the opposite direction of the gradient of the loss score with regard to the weights [25][24][32].

Most visualization methods for the model explanation, especially the one that will be introduced in Section 2.3, are based on forward and backward propagation [34].

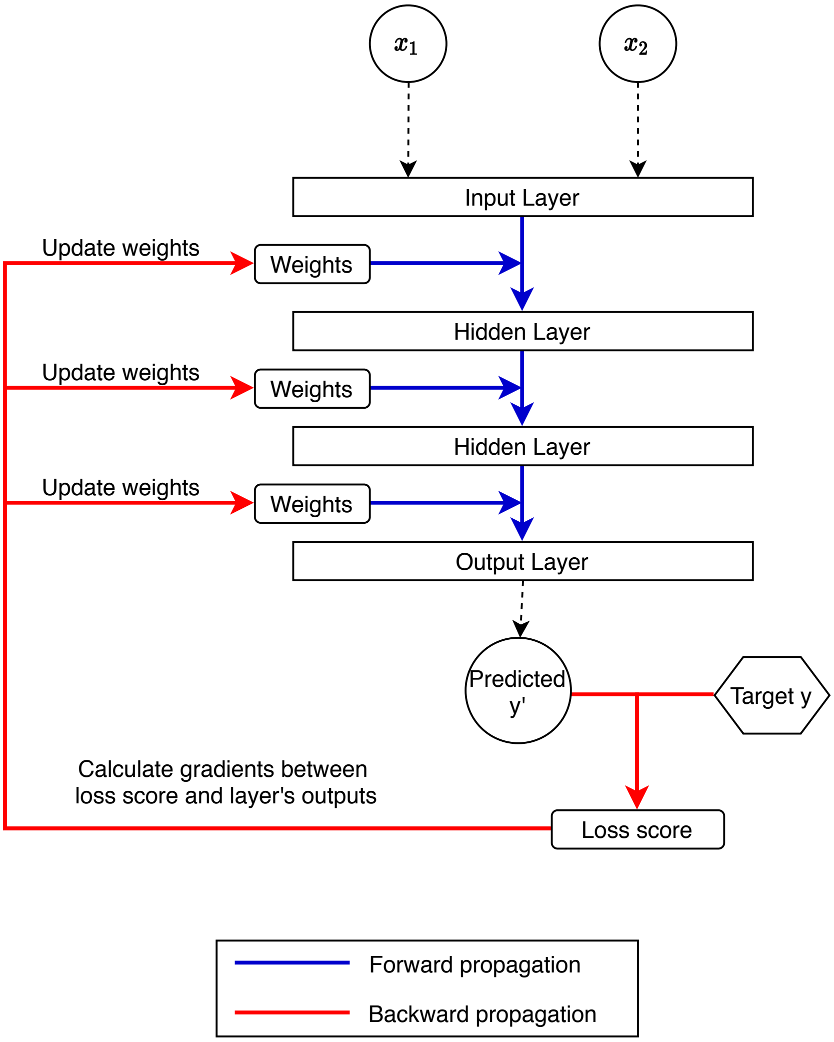


Figure 2.4: Illustration of forward and backward propagation in the neural network. The process of data transformation from the input layer to the output layer is called forward propagation. On the other hand, the process of updating the weights matrices based on the loss score is called backward propagation.

2.2 Convolutional Neural Network

The auto-delineation of cancer tumors is an image-based problem. Since the information extracted from the image data involves the spatial relationship between neighboring pixels, *convolutional* layers [32] are essential in the neural networks that work with image data. A neural network containing convolution operations is called a *convolutional neural network* (CNN) [24][32].

2.2.1 Images and Tensors

The term *Image* in this thesis refers to *Digital Image*, which is composed of **pic**ture **el**ements, called *pixels*. Each pixel contains a numerical value representing the gray intensity at the specific position of the image. For some images, there can be multiple values to represent the intensity at the particular pixel. These sets of values are called *channels* of the images.

When working with image data, we work with the pixel values across the width, height and channels of the images. For 3D images, this data can be more complicated with width, height, depth and channels. In this case, pixels are known as voxels (**vo**lume **el**ements) [35].

Since the image-based data always contains data across multiple dimensions, the term *tensor*, which is a multi-dimensional vector space, is usually used when referring to the image data. A tensor is defined by its *rank* and *shape* [24]. The rank of a tensor is the number of dimensions of the tensor. For example, a tensor with the rank of three, or a 3D tensor, can represent a 2D image with height, width and channels. The shape of a tensor is the size of its dimensions. For an image with a height of 30 pixels, a width of 40 pixels and three channels, it will be represented by a tensor with the shape of (30, 40, 3).

2.2.2 Filter operations

Before defining convolutional layers, we introduce the *filter operations* [35] in digital image processing. Filter operations are usually used for image smoothing, image sharpening, edge and object detection, etc [35]. Filter operations take an image as the input then generate a new image. Each pixel in the new image is calculated individually using the following process. First, based on the coordinates i, j of the new pixel $Out_{i,j}$, a region from the original image Im, denoted as $R_{i,j}$, is taken out using a function $r(Im, i, j)$. This region is called the filter region [35]. After that, the value of new pixel $Out_{i,j}$ is calculated by applying the filter function f on the set of pixels in the filter region. Since the filter region is selected based on the newly created pixel's geometric properties, we can say that the new image is generated while sliding the filter across the original image's width and height.

The relationship between the input image Im and the output image Out can be denoted as:

$$\begin{aligned} Out &= filter\_operaion(Im) \\ \Big[Out_{i,j}\Big]_{m' \times n'} &= \Big[f(R_{i,j})\Big]_{m \times n} \\ &= \Big[f(r(Im, i, j))\Big]_{m \times n} \end{aligned}$$

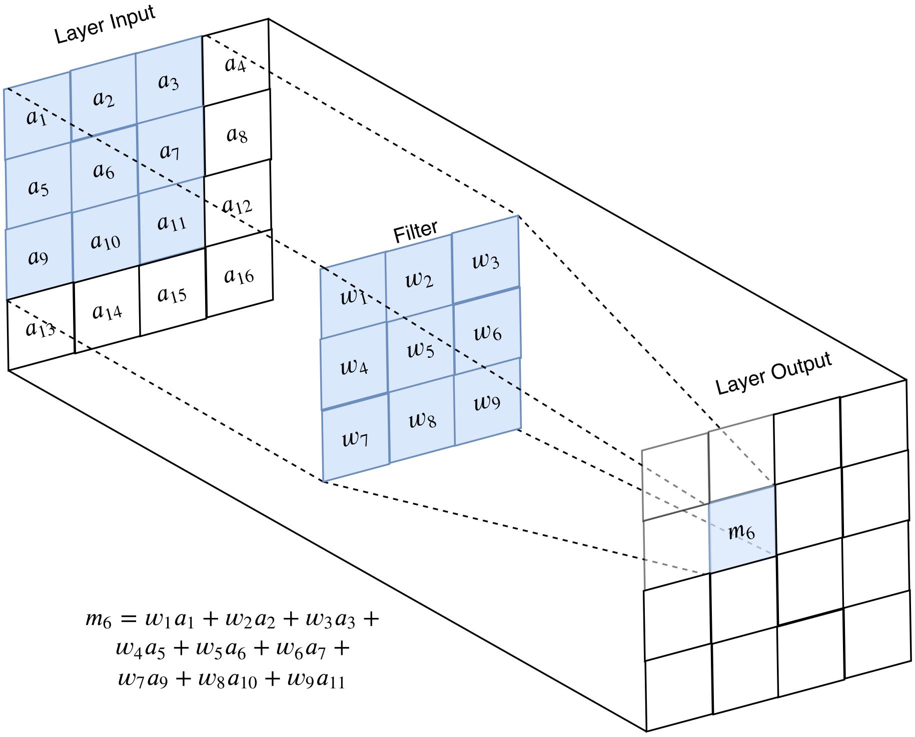
where Im and Out are the original and output images,
$Out_{i,j}$ is the pixel at coordinate i, j of the output image,
$R_{i,j}$ is the filter region of the associated pixels $Out_{i,j}$,
r is the function of selecting the filter region,
f is the filter function.

Depending on the mathematical properties of the function f, filters operations are classified into *linear* and *non-linear* filter operations [35]. While non-linear filter operations are mostly used for noise removal, linear filter operations are usually used for feature enhancement and feature extraction [32][35], which is exactly what we need when working with image data in the neural network.

2.2.3 The convolutional layer

In mathematical terms, the linear filter operation is the convolution operation [24][35]. Data transformation in the convolutional layers is based on convolution operations. Figure 2.5 illustrates the convolutional layer on a 2D image tensor, which uses the convolution operation. In this convolution operation, the filter region is a square of size 3x3, or a 2D tensor of size 3x3. The filter function f uses a filter (aka kernel) of size 3x3 to calculate the linear combination of the pixels in the filter region, then generates the pixel values at the associated positions.

Generally, convolutional layers take a tensor as input and output another tensor. The output tensor of the convolutional layers contains the output images of mul-



$$m_6 = w_1 a_1 + w_2 a_2 + w_3 a_3 + \\ w_4 a_5 + w_5 a_6 + w_6 a_7 + \\ w_7 a_9 + w_8 a_{10} + w_9 a_{11}$$

Figure 2.5: Illustration of the convolution operation on a 2D image. A filter of size 3x3 is used with a linear function to calculated the output m_6.

tiple convolution operations from different filters, followed by an activation function. Each output image is generated by sliding a $k \times k$ filter along the input tensor's width and height. As the filter moves, the filter function f calculates the linear combination between the pixels in the filter region using the $k \times k$ filter, which results in the associated pixel in the output image. Since different filters can be used to extract different features of the input image [24][35], the output tensors of the convolutional layer are the set of different features which are extracted from different filters in the convolution operations. In the case of an input tensor with the rank of n, the same process is applied using filters with the same rank.

Besides, data transformation in the convolutional layers is still a linear combination of nodes (pixels) from the inputs. However, at each node, only a few nodes, which are neighboring pixels of the original image, are used in the linear combination while other nodes, which are unrelated pixels, are silenced. As the filters' values directly affect the convolutional layer's outputs, the weights to be trained in convolutional layers are the values of the filters.

An extension of the convolutional layer is the transposed convolutional layer [36]. This kind of layer is the same as convolutional layers, with the convolution operation and the trainable filters' values. While convolutional layers decrease the sizes of the input tensors, transposed convolutional layers increase their sizes. The differences between these two types of layers involve the different ways of adding zeros padding at the edges of the input tensors, or adding strides of zeros between pixels in the input tensors [24][36].

Another type of layer that uses filter operations in CNNs is the pooling layer [36]. While being called "layer" in the neural network, this type of layer is simply applying filter operations to the input tensors with no trainable weights. This is because this type of layer either uses (1) non-linear filter function or (2) linear filter function with constant filter. One good example of the first case is the *max pooling* layer [32], where the filter function finds the maximum pixel value in the filter region. In the second case, we can look at the *average pooling* layer [32], whose filter function calculates the average value of pixels in the filter region. This function is linear, but the filter value cannot be used as weights. This is because the filter values must be constant. For example, the filter values of the 3x3 filter used in average pooling layers must always be $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$.

2.3 Visualization

Until now, we have gone through the fundamental context of deep learning and components of the neural network, as well as the convolutional neural network when working with image data. These are the keys knowledge of visualizing the CNN model for model explanation. In this part, we will introduce several methods of model visualization, all of which are based on the forward and backward propagation through convolutional layers.

2.3.1 Activation Map

As explained in Section 2.2.3, the outputs of the convolutional layers contain different features extracted from the layer's input. Since each convolutional operation in these layers is followed by an activation function, the outputs of the convolutional layers are called *activation maps* [24]. The activation maps show how the original input images transform as they go through each layer in the convolutional neural network. From the activation maps, we can see which features of the images each layer extracts during the data transformation process.

Figure 2.6 explains how activation maps are generated from a trained neural network. As the process only involves data transformation in the neural network, generating activation maps is the same as the forward propagation of an image through the convolutional layers. Examples of activation maps from the VGG16 model on the ImageNet dataset[3] are showed in Figure 2.7. From an image of a cat, the activation maps show features extracted from the image, such as the foreground (the whole cat), the background, grasses, soils, cat's eyes and ears, and even its edge, etc.

---

[3] VGG16 model is a convolutional neural network with 16 trainable layers proposed by Simonyan and Zisserman [37], and has high performance on the ImageNet dataset [38]. The detailed information about this model can be found in Appendix B.

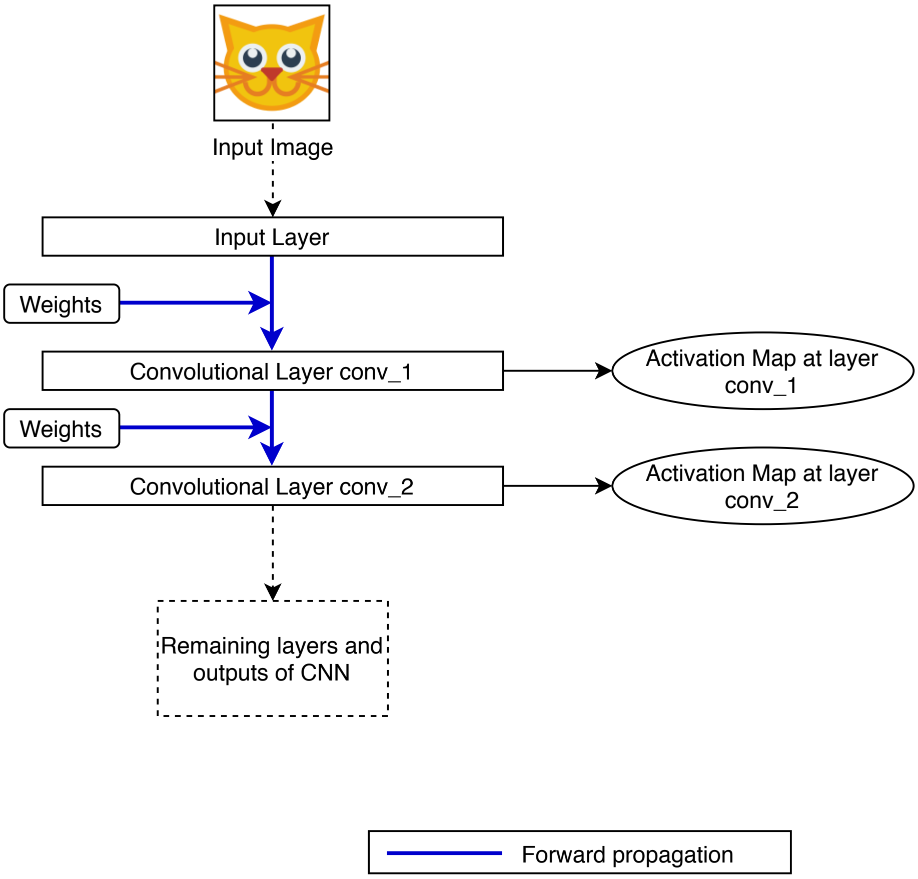

Input Image
Input Layer
Weights
Convolutional Layer conv_1
Activation Map at layer conv_1
Weights
Convolutional Layer conv_2
Activation Map at layer conv_2
Remaining layers and outputs of CNN
Forward propagation

Figure 2.6: Illustration of the process of generating activation maps. From the trained convolutional neural network, the outputs of the convolutional layers are called activation maps, which show the extracted features from the layers' inputs. The process of generating the activation maps is the process of forward propagating an image through the convolutional layers.

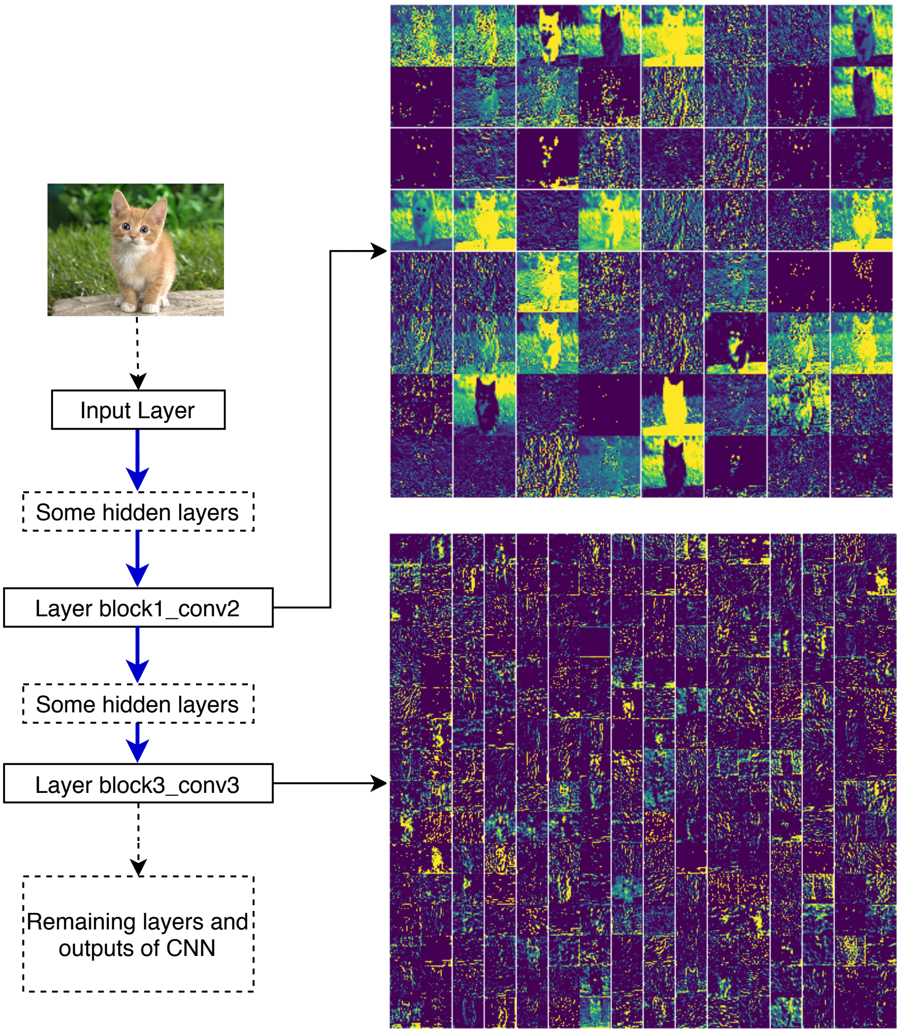


Figure 2.7: Illustration of an example of activation maps from the VGG16 model. From an image of a cat, features in that images are extracted in the layers of the CNN. The features extracted include the background and foreground of the image, parts of the cat such as eyes and ears etc. Images generated by the framework introduced in Chapter 3 on page 29.

2.3.2 Activation Maximization

Erhan *et al.* [15] introduced a method of creating an image that maximizes one or more activation units at specific layers in the convolutional neural networks, called *activation maximization* [15]. Figure 2.8 explains the process of activation maximization. First, from a trained convolutional neural network, the activation maps at a specific layer of an image with random noise I_R is generated. Then, depending on the group of activation units to be maximized, a loss score is calculated based on the generated activation maps. In this case, the loss score represents the magnitude of the required activation units. After that, the gradients between the loss score and the initial image I_R are calculated. Since the goal of this process is to create an image that maximizes the activation maps, the image I_R now acts as the coefficient value in the loss function. Thus, I_R is updated by moving it in the same direction as the calculated gradients. This process is repeated until an image which maximized the required activation units is eventually generated. The resulting image contains the features that each filter in that layer extracts.

From Figure 2.8, we can see that this visualization method is similar to the process of training a neural network, which contains the repetition of forward and backward propagation (see Figure 2.4). The main differences are because of the different goals, leading to the change of loss function, along with the way the coefficients of the loss function are updated in activation maximization.

Example results of activation maximization for filters in the *block5_conv1* layer of the VGG16 model (Table B.1) are shown in Figure 2.9. As the process of forward and backward propagation is repeated several times, the initial image has transformed into some interesting patterns, either eye-like, feather-like, scale-like or even bell-like. From the results, we can conclude that these filters are looking for similar patterns from the input images.

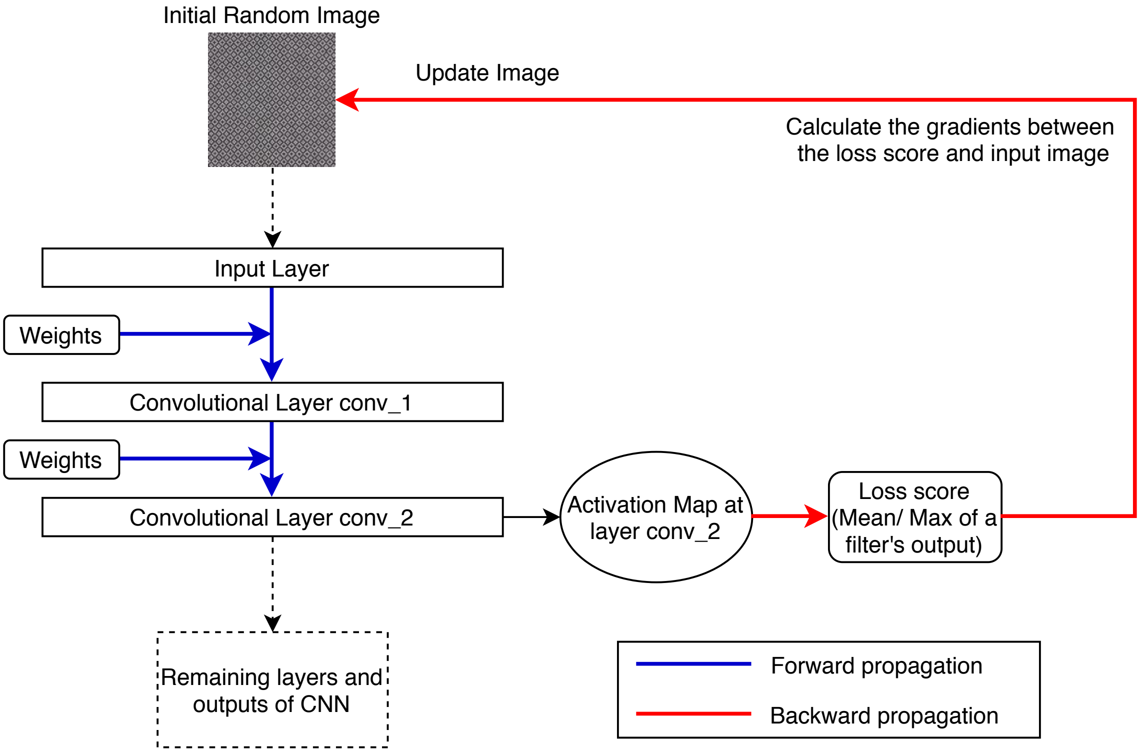
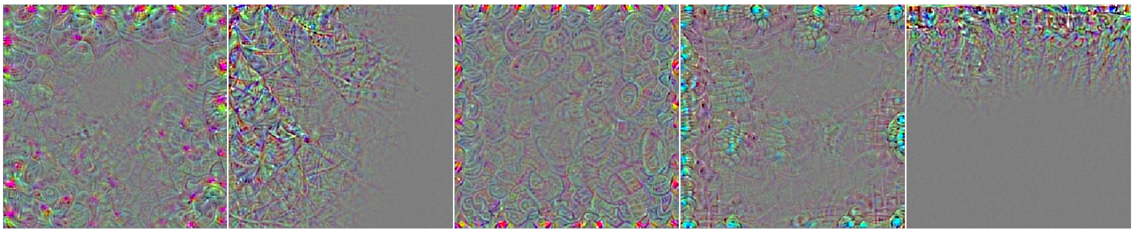
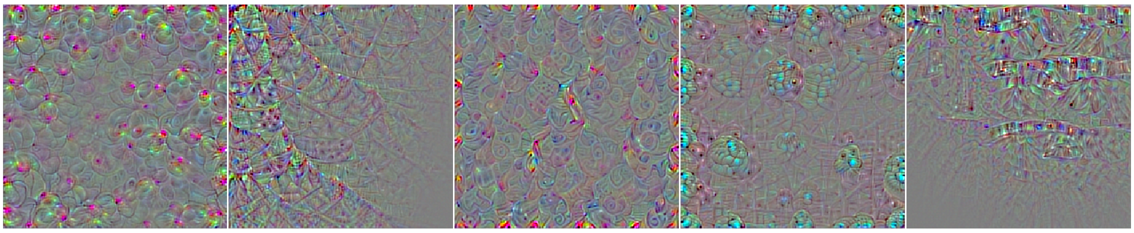

Initial Random Image
Update Image
Calculate the gradients between the loss score and input image
Input Layer
Weights
Convolutional Layer conv_1
Weights
Convolutional Layer conv_2
Activation Map at layer conv_2
Loss score (Mean/ Max of a filter's output)
Remaining layers and outputs of CNN
Forward propagation
Backward propagation

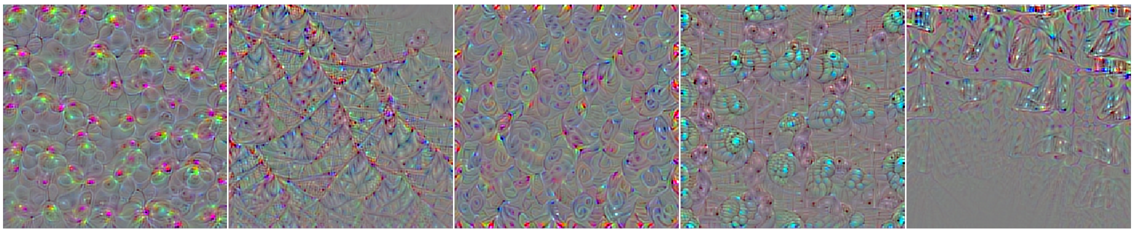
Figure 2.8: Illustration of the process of activation maximization. From the trained convolutional neural network, an image that maximizes that activation maps at a specific layer is generated by repeatedly forward propagating an image (initially with random noise) in the neural network, then use backpropagation to update that image in the direction that will maximize the activation maps.



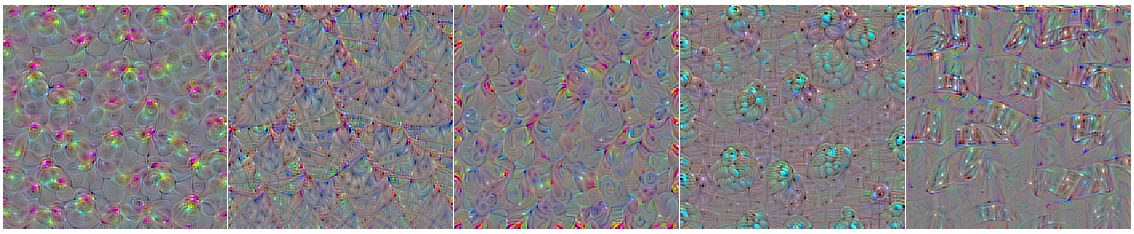
(a) After 5 iterations



(b) After 10 iterations



(c) After 20 iterations



(d) After 50 iterations

Figure 2.9: Example of activation maximization of the VGG16 model. How the initial images with random noise change after 5, 10, 20, 50 iterations (a-d) are shown. Eventually, the images that maximize filters in the *block5_conv1* layer are generated. Images generated by the framework introduced in Chapter 3 on page 29.

2.3.3 Saliency Map, Deconvnet and Guided Backpropagation

Saliency Map [18], *Deconvnet* [19] and *Guided Backpropagation* [20] are the visualization methods with the goal of finding parts of the input image which are the most important for the output of the convolutional neural network. Because the CNN model's output is the result of a differentiable function on the input image and the weights in that CNN model, the gradients of the model's output with respect to the input image show which pixels of the input image have the most impacts on the prediction of the CNN model. The three visualization methods are proposed based on this idea. Although the initial proposals of these methods are used on class probability results in the CNN model, we can also apply these three methods on any layers in the convolutional neural network.

Figure 2.10 explains how these three methods are applied to a specific layer in the convolutional neural network. First, an input image is propagated forward in a trained CNN. After that, a backpropagation step is performed. From the activation maps, a loss score, depending on the goal of the process, is calculated. For a classification model, the loss score is the class probability of the image predicted by the model. In a more general case, the loss score is the value calculated from the nodes we want to analyze in the neural network. From the calculated loss score, its gradients with respect to the input are calculated, resulting in an image acting as a heatmap of the importance of each pixel to the output of the model (or layers).

The only difference between these three methods is the way the gradients are calculated, which will be explained in the following parts.

Saliency Map

The saliency map [18], or the gradient map, was first introduced by Simonyan *et al.* in 2013 [18]. In this method, the resulting image is generated by directly differentiating the loss score with respect to the input image. Since this result is the actual gradients between the loss score and the input image, the resulting image is quite noisy (Figure 2.11).

The process of calculating the saliency map at the i^{th} layer $f^i_{saliency}$ can be denoted

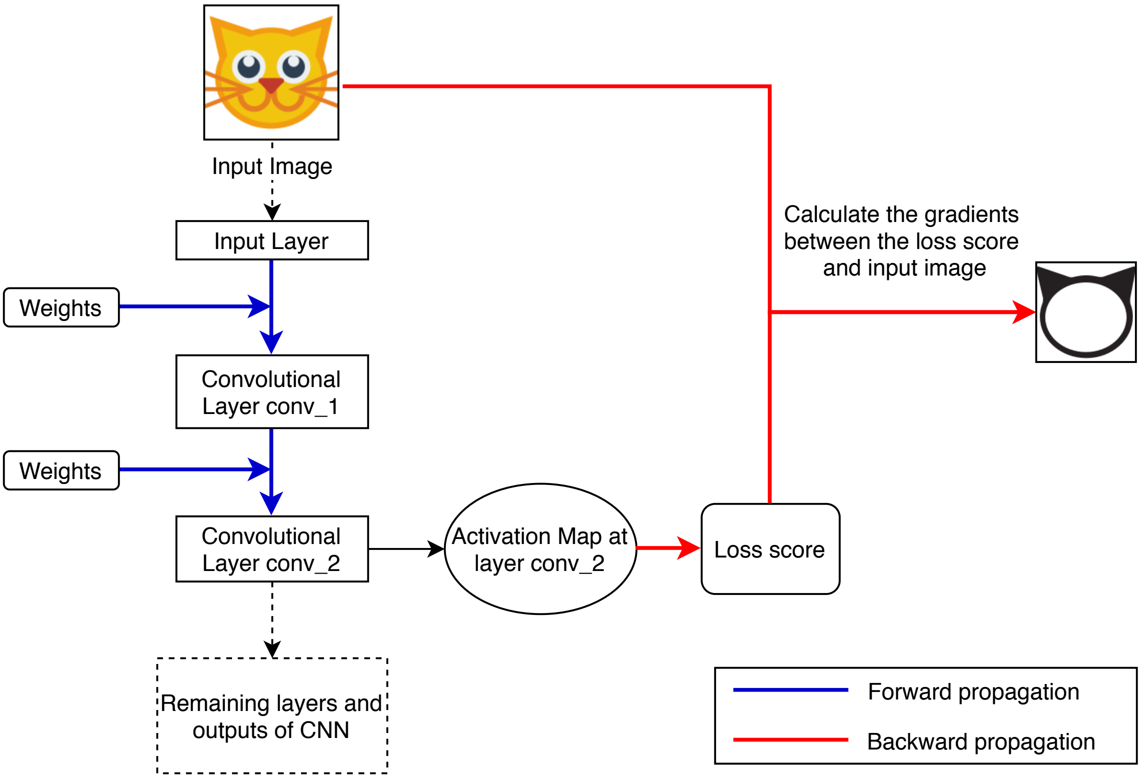

Input Image
Input Layer
Weights
Convolutional Layer conv_1
Weights
Convolutional Layer conv_2
Activation Map at layer conv_2
Loss score
Remaining layers and outputs of CNN
Calculate the gradients between the loss score and input image
Forward propagation
Backward propagation

Figure 2.10: Illustration of the process of generating saliency maps, deconvnet and guided backpropagation. From the trained convolutional neural network, an image is propagated forwardly. After that, a loss score is calculated. The pixels that have the most impacts on the loss score are calculated using the gradients of the loss score with respect to the input image.

as:

$$
\begin{aligned}
f^i_{saliency} &= \frac{\partial I_{out}}{\partial I_{in}} \\
&= \frac{\partial L^i_{out}}{\partial L^i_{in}} \cdot \frac{\partial L^{i-1}_{out}}{\partial L^{i-1}_{in}} \cdot \ldots \cdot \frac{\partial L^1_{out}}{\partial L^1_{in}} \\
\text{for } I_{out} &= loss\_fn((f^i \circ f^{i-1} \circ \ldots \circ f^1)(I_{in})) \text{ and} \\
F^i_{out} &= f^i(F^i_{in})
\end{aligned}
$$

where I_{out} is the loss score, I_{in} is the input image, $loss\_fn$ is the loss function and f^i is the function mapping the input F^i_{in} and output F^i_{out} of the i^{th} layer.

Deconvnet

The deconvnet method, which was introduced by Zeiler and Fergus, also calculated the gradients of the loss score with the input image. However, this method is slightly different from the saliency map of Simonyan *et al.* [18] as the gradients when backpropagating through the ReLu activation function are calculated differently. When backpropagating the gradients using the chain rule, instead of calculating the actual gradient of the ReLu function, the deconvnet method applies the ReLu function on the gradients being backpropagating. This means that the deconvnet results only focus on pixels that have positive impacts on the output.

The following denotes how saliency maps and deconvnet are different when handling the ReLu function in backpropagation:

$$\text{For } F^i_{out} = f^i(F^i_{in}) = \phi_{relu}(conv(F^i_{in})) = \phi_{relu}(C^i) \text{ and } R^{i+1} = \frac{\partial L^{i+1}_{out}}{\partial L^{i+1}_{in}} \text{ ,}$$
$$R^i_{saliency} = (C^i > 0) \cdot R^{i+1}$$
$$R^i_{deconvnet} = (R^{i+1} > 0) \cdot R^{i+1}$$

Due to the change of gradient calculation, the deconvnet method gives a less noisy result than the saliency methods [20] (Figure 2.11). However, in deeper layers in the CNN, the deconvnet method is unable to give a sharp and recognizable image [20], which leads to the proposal of Springenberg *et al.* [20], the guided backpropagation method.

Guided Back-propagation

Springenberg *et al.* [20] proposed a visualization method that combines both the saliency map and the deconvnet method. In guided backpropagation, when backpropagating through the ReLu activation function, the gradient of the ReLu function are still calculated. However, the ReLu function is still applied tp the gradients being backpropagating. Therefore, the negative signals in both forward and backward propagation are zeroed out, resulting in an image with sharper lines and features (Figure 2.11).

The following shows how the gradients through the ReLu function are calculated in guided backpropagation.

$$R^i_{guided\_backpropagation} = (C^i > 0) \cdot (R^{i+1} > 0) \cdot R^{i+1}$$

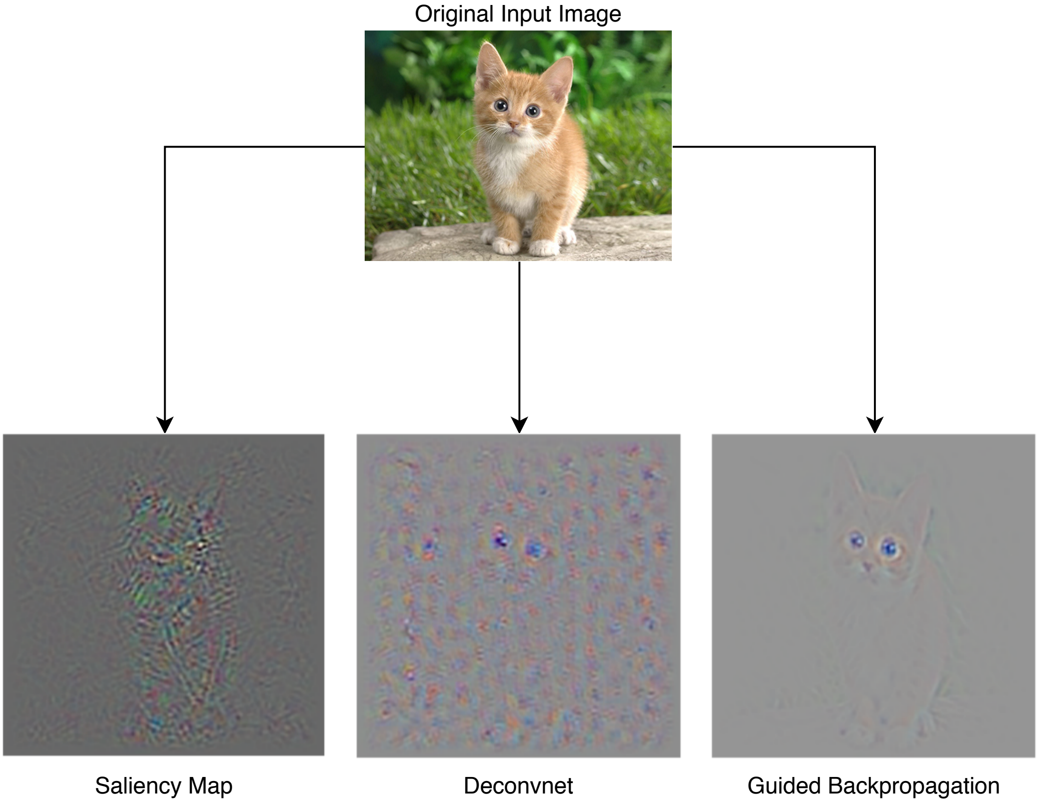


Figure 2.11: Example results of saliency map, deconvnet and guided backpropagation. The results of saliency map, deconvnet and guided backpropagation are generated by Springenberg *et al.* [20] using a model trained on the ImageNet dataset [38]. In these results, parts of the input image that influence most to the results of the model are highlighted. Unlike the saliency map with a noisy image, deconvnet gives a clearer image with less noise while guided backpropagation generates an image with sharper lines and colors.

Chapter 3

Code

This chapter provides an overview of the deep learning framework developed in conjunction with this Master's thesis and the possible updates implemented to support model visualization. This chapter also describes the quality control of the implemented code.

3.1 Deoxys Framework

3.1.1 Introduction and usage

As a preparation for this Master's thesis, the author developed a framework to apply deep learning for tumor segmentation in medical images as part of the coursework in DAT390 Data Science Seminar. This framework, called *deoxys*, allows users to create and train a convolutional neural network on a set of images, as well as visualize the performance of the training process and view the prediction of the trained model (Figure 3.1).

The *deoxys* framework is generalized to work with different forms of image data and CNN architectures. Users can define their CNN using configurable JSON files, which allow them to choose the layers, loss functions, activation functions, performance metrics and other components in neural networks.

Moreover, the *deoxys* framework is also modular so that it is easy to maintain, extend, and update. Its high flexibility allows users with advanced programming background to define their customized components easily. An example of this flexibility property can be found at `https://github.com/yngvem/ntnu-analysis/blob/master/experiment.py`, where the user was able to define his customized loss functions and performance metrics.

The developed framework is available at `https://github.com/huynhngoc/deoxys`. In addition, the detailed information about the *deoxys* framework can be found in Appendix A on page 107

3.1.2 Structures

The *deoxys* framework was developed using Keras [23] as the base library. The Keras library provides many implemented CNNs' layers, loss functions, activation functions, etc. The Keras library [23] works as the top-level interface to communicate with other *deep learning backends*[1] such as Tensorflow [39] and Theano [40].

The minimum software requirement for the *deoxys* framework to work properly is Python 3.7 and Keras 2.3.0.

The main components of the framework illustrated in Figure 3.2 are: (1) the data-reader, which reads the image data then feeds them to the model for training and testing; (2) the Keras model [23] wrapper, which is the deep learning model to be trained; (3) loader modules, which are used for loading the configuration file to build the model wrapper; (4) experiments modules, which are used for training models with different hyper-parameters and for visualizing the performance (Figure 3.2).

The data-reader has three usages. The first usage is to read the image data from disk. The second usage is to split the data in training, validation and test set. This makes k-fold cross-validation [41] possible[2]. The final usage is to feed the model with small batches of processed data while training, validating and testing.

The Keras model wrapper contains methods for training and testing the model,

---

[1] These refer to deep learning frameworks which allow efficient data computation in deep learning model by managing memory usages, utilizing the CPU and the GPU of the computing environment.

[2] Splitting up data into different sets for cross-validation is a technique used when training a deep learning model (see Raschka and Mirjalili [25] for details). As these terms are not related to the main goals of this Master's thesis, they will no longer be discussed in this thesis.

and methods for model serialization such as saving and loading a model to and from disks (see Model part in Figure 3.2).

The loader modules, as illustrated in architecture loader part and model objects parts in Figure 3.2, create components of the model from configuration JSON files. These components are either Keras implemented objects (loss functions, activation functions, metrics) or user-defined objects. These modules also contain predefined architectures such as the Sequential architecture [24], which is the simple form of CNNs with a stack of layers, and U-net architecture [42], which is a more complicated architecture used for segmentation problems.

Experiment modules allow users to train different CNNs with different components and architectures, where the performance logging and the model serialization are applied in every iteration (see single and multiple experiment parts in Figure 3.2). Thus, other usages of these modules are to visualize the performance of an experiment, and to visualize the predictions of some samples in the validation and test set of the dataset.

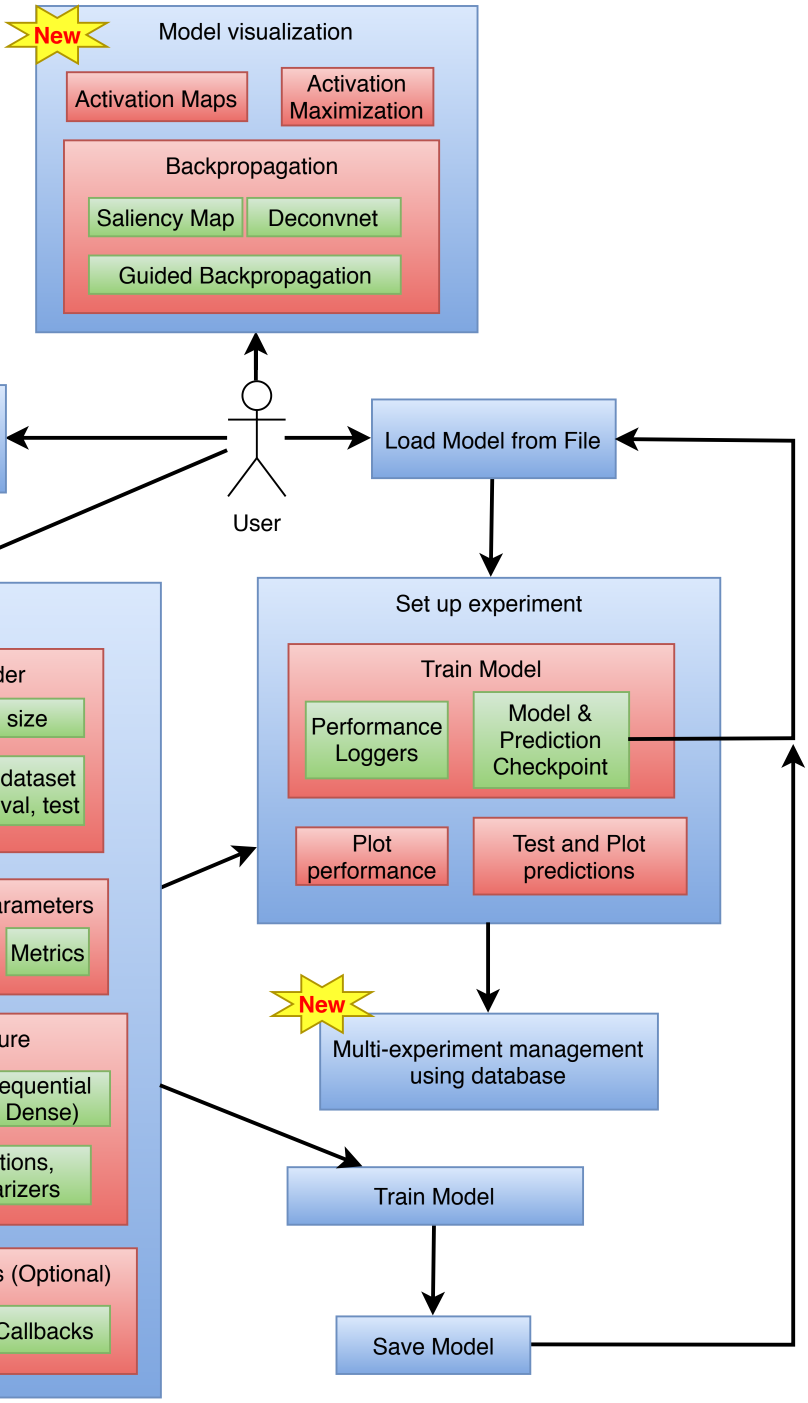


Figure 3.1: Use case diagram of the *deoxys* framework developed by the author as part of the coursework in DAT390 Data Science Seminar (details in Appendix A). This figure shows how the users interact with the *deoxys* framework. Use cases marked with the "new" label are the newly added use cases in this Master's thesis.

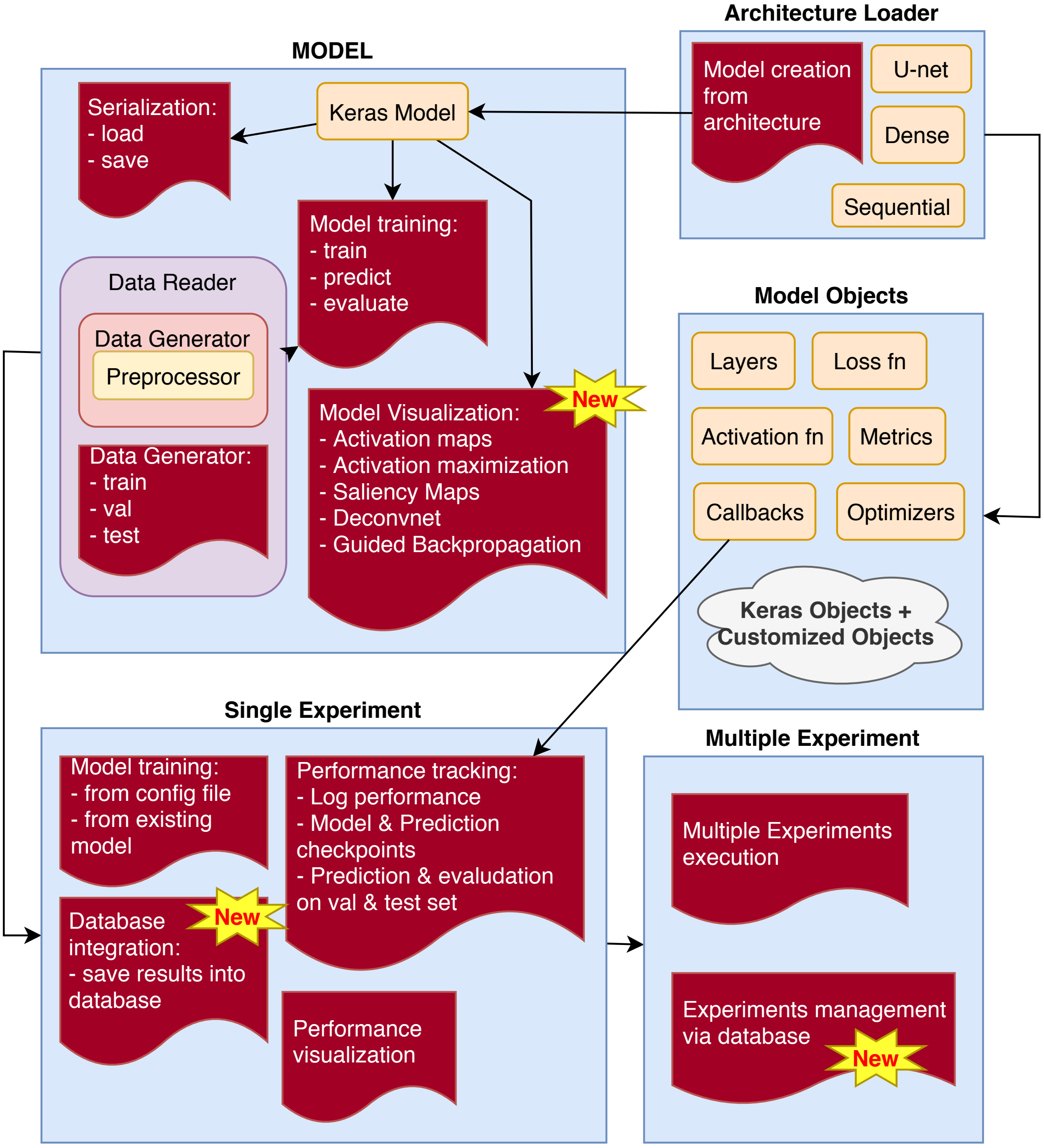


Figure 3.2: Structure of the *deoxys* framework developed by the author as part of the coursework in DAT390 Data Science Seminar (details in Appendix A). This figure shows the components of the framework, as well as the available groups of functions. Parts marked with the “new” label are updates of the *deoxys* framework in this Master’s thesis.

3.2 Updates

One of the goals of this Master's thesis was to implement supports for visualization of the model features in the *deoxys* framework. This section describes the updates added to this framework. These updates include the implementations of several visualization methods, as well as the management of training models using a database. Figures 3.1 and 3.2 show the new software features developed in this Master's thesis.

3.2.1 New APIs for model visualization

The main focus of these updates is the integration of model visualization methods discussed in Section 2.3 on page 18 into the Keras model wrapper. These methods include activation maps [24], activation maximization [15], saliency maps [18], deconvnet [19] and guided backpropagation [20].

Users of the *deoxys* framework should be able to visualize the deep learning model by using the following APIs.

deoxys.model.Model.activation_map

This function should take a layer name and a list of images as input arguments, then generate the associated activation maps at that layer of these images. Users can use this function to see how the input images changes in the intermediate layers in the convolutional neural network.

deoxys.model.Model.activation_maximization

Users can easily find the patterns, or features that a specific layer in the convolutional neural network extracts using this function. In addition, users should have the option to choose the initial image and the loss score function, as well as the number of steps to repeat the process in Figure 2.8 on page 22.

deoxys.model.Model.backprop

Users can view the influence of each part of the image on the prediction of the model using this function, which implements the saliency maps method. For complex deep learning problems, users should have the option to define their customized loss function in Figure 2.10 on page 25.

deoxys.model.Model.deconv

Similar to the previous function, aspects that affect the decision of the model can be found using this function, which gives the implementation of the deconvnet method.

deoxys.model.Model.guided_backprop

This function is similar to the two previous functions, but guided backpropagation method is implemented in this function.

3.2.2 Quality Control

The implementations of activation maps and activation maximization using Keras were based on example codes [24] which have been used across numerous examples in the GitHub community[3]. Therefore, the implementations of these two methods have a high level of reliability. On the other hand, visualization methods that depend on the modification of gradient calculation in backpropagation did not have Keras-based examples. If examples were available, there were based on an old version of Keras. Therefore, to ensure that the implementation of these methods in *deoxys* was reliable, we needed to compare the results provided by the *deoxys* framework to previously published results. The visualization results by Springenberg *et al.* [20] (see Figure 2.11 on page 27) using the VGG16 model on ImageNet data (Appendix B) were chosen as a comparison for saliency maps, deconvnet and guided backpropagation visualization given by *deoxys*.

---

[3] GitHub (`github.com`) is one of the places where software developers share their ideas and codes for solving different problems.

3.2.3 Others extension

Another update of the *deoxys* framework (Figures 3.1 and 3.2) was the integration of a database to manage the experiments. When searching for the best CNN model, users have to run different experiments multiple times to get enough data to analyze the performance logs. Analyzing different log data requires time and skills to combine these data as they stay at different locations in the computers. Therefore, users need a system to centralize and organize the data, which leads to the need for a database management system (DBMS).

Database integration allowed the *deoxys* framework to communicate with a DBMS to manage the results from multiple experiments. The DBMS manages these following five types of objects: (1) the experiments, (2) the sessions, (3) the saved models, (4) the saved predictions and (5) the performance logs. The relationship between these types of objects is illustrated in Figure 3.3.

Each type of object in the DBMS is called a *table*, where the name of the table is the type name. In a table, each type of information about the objects is called a *column* or a *field*. In the DBMS, objects of one type are distinguished using an *id*. Thus, each table contains an *id* column (Figure 3.3).

The DBMS manages different experiments, each of which contains the user-defined name and description, as well as the configuration information which defines the architecture and hyper-parameters used in this experiment (see the *experiment* table in Figure 3.3).

Each run of an experiment is called a *session*. Each session contains the information about the experiment it belongs to, when it is created and last modified, how many epochs it has run, and its status (whether it just created, is running or has finished) (*session* table in Figure 3.3). In each session, there are different models and different predictions saved at some epochs. These objects are also managed by the DBMS (see *models* and *predictions* table in Figure 3.3). In addition, the performance results at every epoch of a session are also stored in the DBMS in the *perf_log* table. Apart from the information about the session and epoch number each performance result belongs to, other fields in this table are indeterminate. Because users can define different performance metrics in different experiments, sessions from different experiments have different performance results, resulting in different fields in the *perf_log* table.

Because of the indeterminacy in the fields in the *perf_log* table, MongoDB[4], a

---

[4] Official website at https://www.mongodb.com/.

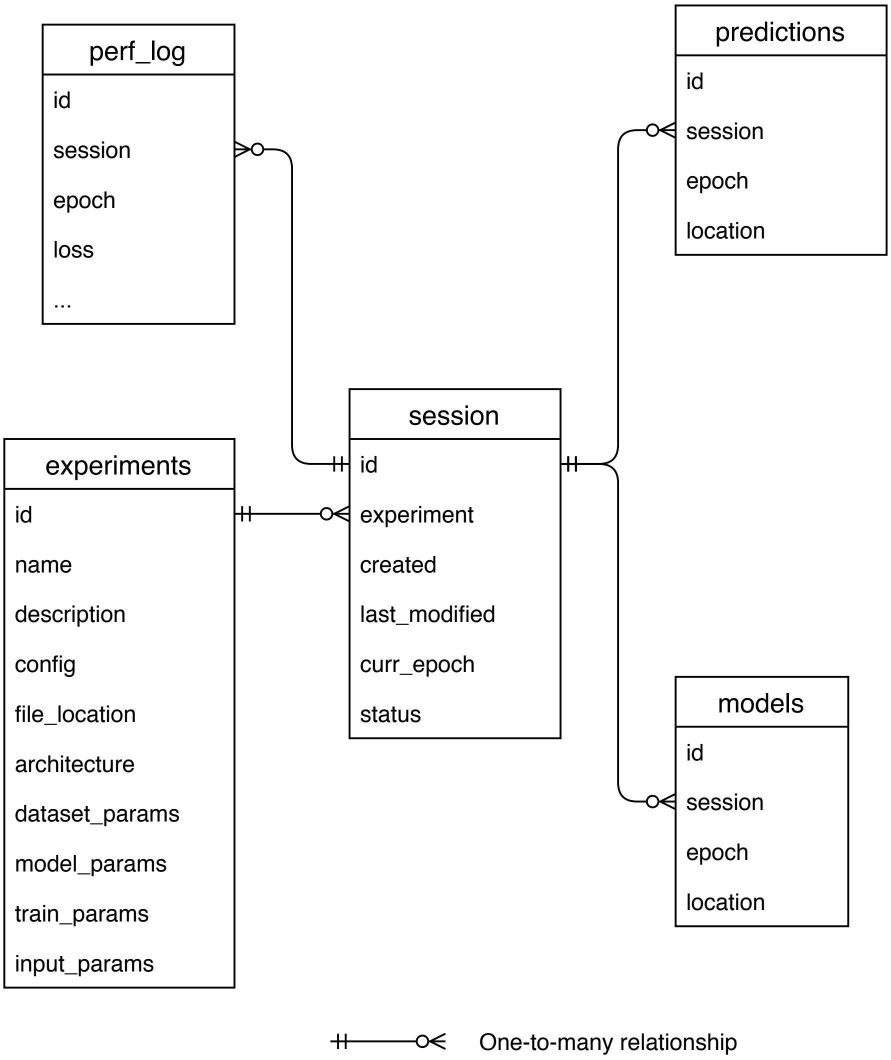


Figure 3.3: The relationship between different types of objects in the database. Each box represents a table in the DBMS. The tables' names and fields are separated by horizontal lines in these boxes. In this figure, each experiment contains many sessions. Each session contains different performance results, saved models and saved predictions at different epochs (iteration).

Name	Description
DBMS-UI	The database management system's user interface. Users can view the data stored in the DBMS using this interface. Usually, the DBMS is provided together with a user interface.
The sample config	The JSON file containing the configuration of a sequential CNN model to classify images of hand-written digits into digits.

Table 3.1: Materials for test cases.

Test case	Step	Description	Expected Results	Results
Add an experiment	1	Clear all data in the DBMS	In the DBMS-UI, the *experiment* table has a new item with the name of "Test 01".	Passed/ Failed
	2	Create an experiment from the sample config with name "Test 01" and empty description		
	3	Check the DBMS-UI		

Table 3.2: Example of a test case in database integration.

database management system with high performance on dynamic data [43][44], was chosen to be integrated into the *deoxys* framework. Nevertheless, for flexibility, maintainability and updateability of the *deoxys* framework, the integration was designed to make it possible to interact with any other types of DBMSs in the future.

To ensure the reliability when integrating the database into the *deoxys* framework, seven manual test cases were written and tested using the materials defined in Table 3.1. An example of a test case can be found in Table 3.2. All test cases are available at the full test report in Appendix C on page 123.

Chapter 4

Experimental setup

Model visualization methods implemented in this thesis were tested on a pretrained deep learning model for segmentation of head and neck cancer tumors. This chapter describes the trained model and dataset that were used in this thesis. In addition, this chapter also describes how the visualization methods were applied to this model for interpretation.

4.1 The dataset and the pretrained model

The dataset and the pretrained model used in this Master's thesis replicated the segmentation model with the highest performance developed in 'Deep learning for automatic delineation of tumours from PET/CT images' by Moe [11]. This model was implemented and trained in the newly developed *deoxys* framework using the same layers and hyper-parameters.

Note that the replication was only limited to the model and the dataset for reproducibility, while the *deoxys* framework is a newly developed framework. This framework contains several modules for defining and customizing deep learning models, visualizing the performance, automatically finding the best model within an experiment or among several experiments, supporting K-fold cross-validation [41], which were not included in [11]. Note also that *deoxys* is a complete framework encompassing all elements required for deep learning model development (see Figure 3.2 on page 33). The framework is general and not limited to tumor segmentation and enables the use of a range of different deep learning architectures,

Dataset	No. patients
train	142
val	15
test	40

Table 4.1: The number of patients in each of the datasets. See [11] for details

loss functions, activation functions, performance metrics, etc.

4.1.1 Head and neck cancer dataset

3D PET/CT images and delineation masks from 197 patients with head and neck cancer at the Oslo University Hospital, the Radium Hospital, were used for training. The manual delineation masks provided by the clinicians included pathologic lymph nodes and the gross tumor volume (GTV). When multiple manual delineations existed, the union of these delineations was used as the ground truth (model's true target). 2D slices with two channels, CT and PET of the 3D image of the patients, were used as one item in the dataset. As the dataset was the replication of Moe's thesis [11], this dataset was split as described in Table 4.1, and was stratified by tumor stage. In addition, the Hounsfield windowing preprocessing [11] was performed on the dataset. The detailed description of the dataset can be found in Moe's work [11].

In this Master's thesis, the model visualization methods was applied to the CT/-PET images in the test dataset.

4.1.2 The Model

Unet architecture

The model used to train on the head and neck cancer dataset in Section 4.1.1 had the Unet architecture [45], which is illustrated in Figure 4.1. This architecture contains a down-sampling path using Max Pooling layers to decrease the spatial dimension of the layers' outputs and an up-sampling path using Transposed Convolutional layers to increase the spatial dimension of the layers' outputs. These two paths are connected by a bottle neck path at the bottom of the U-shape of the architecture.

This model was trained for 14 epochs, with a low learning rate[1] of 10^{-4} using the Adam optimizer[2] [46].

Detailed information about layers in this model and their relationship can be found in Tables D.1 to D.4.

Model performance

Since the cancer tumor delineation is a segmentation problem, the Dice score[3] [47] was used as a metric for analyzing the performance. This type of metric is calculated based on the number of true positive ("tumor" pixels predicted as "tumor"), false positive ("non-tumor" pixels predicted as "tumor") and false negative ("tumor" pixels predicted as "non-tumor") in the predicted images.

With true positive denoted as TP, false positive denoted as FP, and false negative denoted as FN, the Dice score is calculated based on the following equation:

$$dice = \frac{2TP}{2TP + FN + FP}$$

When evaluating the trained model on the test dataset, we got the result of 0.646 for the average Dice score over the entire dataset. The median of these Dice scores was 0.779. When merging all slices of each patient then calculating the Dice score on all pixels in the resulting 3D images, we got an average Dice score of 0.733 per patient and the median of 0.775. These results concur with the results obtained by Moe [11] of an average Dice score of 0.66 with the median Dice score of 0.75 and a per-patient Dice score of 0.65 with the median Dice score of 0.67. Thus, the *deoxys* framework was capable of replicating the results of Moe [11].

---

[1] In Section 2.1.4, the weights of the neural network are modified based on the gradients of the loss score with respect to the weights. Learning rate is the magnitude of the changed values when adjusting the weights.

[2] An optimizer is an algorithm for minimizing the loss score defined in Section 2.1.3 on page 11 in the backpropagation process (Figure 2.4 on page 13).

[3] Binary F-beta with $\beta = 1$ and f1 score refer to the same term with Dice score.

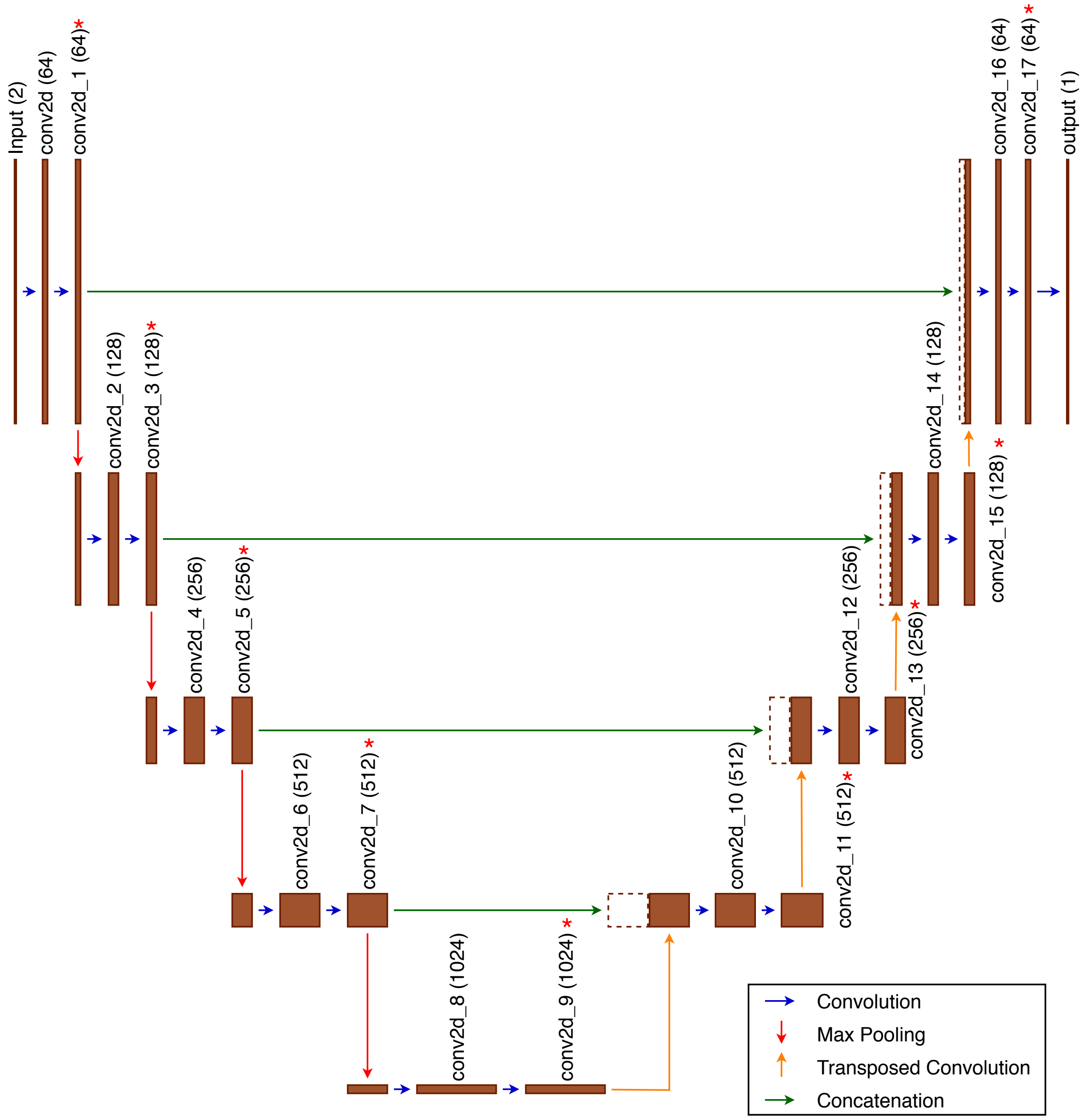


Figure 4.1: Illustration of the Unet architecture of the model used in this Master's thesis. Convolutional layers are marked with name and number of filters used in these layers. The down-sampling path lies between the input layer and the conv2d_8 layer. The path from conv2d_8 to conv2d_9 is the bottleneck path. Finally, the remaining path, from the conv2d_9 layer to the output layer is the up-sampling path. The layers marked with the red asterix (*) were used for model interpretation as listed in Table 4.5.

Patient idx	Slice idx	Dice score
91	86	0.94
148	11	0.94
209	14	0.95
217	20	0.96
233	48	0.96
249	55	0.95

Table 4.2: Information of images with high Dice score for interpretation.

Patient idx	Slice idx	Dice score
60	49	0.52
191	62	0.53
194	71	0.52
217	54	0.53
233	16	0.52

Table 4.3: Information of images with Dice score of 0.5 (intermediate performance) for interpretation.

4.2 Visualization

The images used for model visualization were from the patients in the test set of the head and neck cancer dataset. We focused on three groups of image data based on the Dice score in this set. The first group comprised six images with high Dice score, as listed in Table 4.2. The second group comprised five images with Dice score of 0.5, as listed in Table 4.3. The final group comprised five images with low Dice score (Dice score of 0, i.e. completely wrong delineation, no overlap between the clinicians and model's delineation), as listed in Table 4.4.

Since the pretrained model was deep and complex, we also focused on some specific layers, which are the convolutional layers immediately before the max pooling layers, convolutional layers before transposed convolutional layers and the layer before the output layer in Figure 4.1. These layers' names and number of filters can be found in Table 4.5.

Patient idx	Slice idx	Dice score
8	155	0.00
16	35	0.00
110	1	0.00
148	76	0.00
249	19	0.00

Table 4.4: Information of images with Dice score of 0.0 for interpretation, indicating that the model delineations for these slices did not overlap with the ground truth.

Layer name	No. Filters
conv2d_1	64
conv2d_3	128
conv2d_5	256
conv2d_7	512
conv2d_9	1024
conv2d_11	512
conv2d_13	256
conv2d_15	128
conv2d_17	64

Table 4.5: Layers in the Unet architecture (Figure 4.1) used for model interpretation.

4.2.1 Activation maps

Because deep learning models are often compared to indecipherable black-boxes, activation maps of image slices with high Dice scores as listed in Table 4.2 were generated to see what occurs in the intermediate layers of the model. Due to the complexity of the pretrained Unet model (Figure 4.1), we only generated the activation maps of these images at the layers listed in Table 4.5.

4.2.2 Activation Maximization

To obtain a deeper understanding of features that layers in Table 4.5 extracted, the activation maximization method was applied on these layers to generate the patterns that these layers search for (this process is explained in Section 2.3 on page 18). Note that the activation maximization method used in this section did not involve or depend on any images from the head and neck dataset, as this process starts from an image with random noise (see Figure 2.8 on page 22).

4.2.3 Gradient-based visualization

To find which pixels in the input images influenced the prediction of the model, saliency maps, deconvnet and guided backpropagation were applied to three groups of data listed in Tables 4.2 to 4.4. In addition, the author defined two different *loss functions* and used them in the process illustrated in Figure 2.10.

The first loss function was the total confident values[4] of pixels predicted as "cancer tumor". Hereafter, this loss function will be referred as the *positive prediction* loss function. The images generated using this *positive prediction* loss function showed the pixel importance for the segmentation results. If the output tensor is denoted as $\hat{Y}$, and each value in this tensor is denoted as $\hat{y}_{ij}$ where i and j are the coordinates of this value, this loss function $J_{positive\_prediction}(\hat{Y})$ in this case is calculated based on the following equation:

$$J_{positive\_prediction}(\hat{Y}) = \sum \left(\hat{y}_{ij} \cdot f_{threshold}(\hat{y}_{ij})\right)$$

$$\text{where } f_{threshold}(x) = \begin{cases} 0, & \text{if } a < 0.5 \\ 1, & \text{otherwise} \end{cases}$$

---

[4] Each pixel in the outputs of the described Unet model had a value between 0 and 1 inclusively, which is the probability of being the cancer tumor with a threshold at 0.5.

The other loss function was the total confident values of pixels which were "cancer tumor" in the ground truth. For slices with Dice score larger than 0, this implied the true positive values. For slices with incorrect segmentation (Dice score of 0), the images generated by this loss function would show which pixels should change positively or negatively so that the model could make the right prediction. This loss function will be referred as the *true positive* loss function for later discussion. By using the previously defined notations, in addition to denoting the ground truth tensor as Y, and each value in this tensor is denoted as y_{ij}, the following equation gives this *true positive* loss function.

$$J_{true\_positive}(\hat{Y}) = \sum \left(\hat{y}_{ij} \cdot f_{threshold}(y_{ij})\right)$$

Note that these *loss functions* are not the loss functions used for training the model as described in Section 2.1.3 on page 11. Instead, these *loss functions* are defined depending on the goal of the visualization process (see Figure 2.10 on page 25). In this case, the goals are finding the pixels that contribute to the segmentation result for the *positive prediction* loss function, and finding the pixels that contribute to the correct delineation for the *true positive* loss function.

Chapter 5

Results

This chapter shows how the updates of the *deoxys* framework proposed in Chapter 3 have been implemented, and specifies the results of quality control of the framework. In addition, this chapter also provides the results obtained using the visualization methods for interpretation of the auto-delineation model of the head and neck cancer dataset, as described in Chapter 4.

5.1 Implemented updates of *deoxys*

5.1.1 Visualization

The APIs proposed in Chapter 3 are currently available and accessible in the latest version of the *deoxys* framework.

For activation maps and activation maximization, the implementation using Keras [23] followed the instruction provided by Chollet [24]. In addition, the implementation of the activation maximization method used L2 regularization [48] to normalize the gradients calculated.

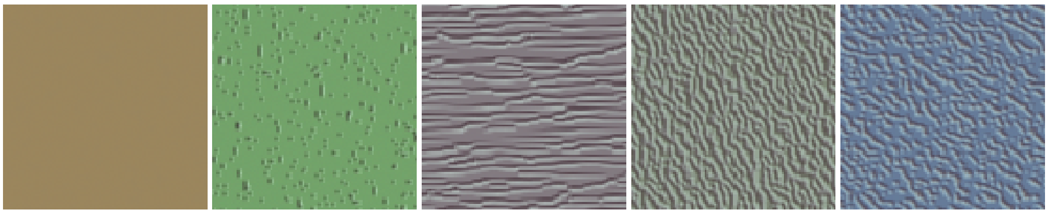
Figure 5.1 shows the results of generating the activation maximization of several layers in the VGG16 model trained on the ImageNet dataset [37] using the *deoxys* framework. After 50 iterations, we can see that the filters in the first convolutional layer (the block1_conv1 layer in Figure 5.1a) extracts colors and edges. The filters in block2_conv1 and block3_conv1 layer in Figure 5.1b and c extracts some combination of colors and patterns (stripes and dots). Filters in higher layers (block4_conv1 and block5_conv1 layers in Figure 5.1d and e) extracts textures that can be found in the real world, such as feathers, eyes, nets, scales, etc. Note that only a few lines of code are needed to generate the images in the activation maximization process (see below lines 11 to 15).

```
1  # Import necessary libraries
2  from deoxys.model import Model
3  from tensorflow.Keras.applications import vgg16
4  
5  # Load the pre-trained VGG16 model
6  vgg = vgg16.VGG16(weights='imagenet', include_top=True)
7  # Put the pre-trained model into deoxys's Model
8  deo = Model(vgg)
9  
10 # Generate image that maximizes first filter of block5_conv1 layer
11 max_filter = deo.activation_maximization(
12     'block5_conv1', # Layer's name
13     epochs = 50, # Number of iterations
14     filter_index = 0 # Filter's index (starts from 0)
15 )
```

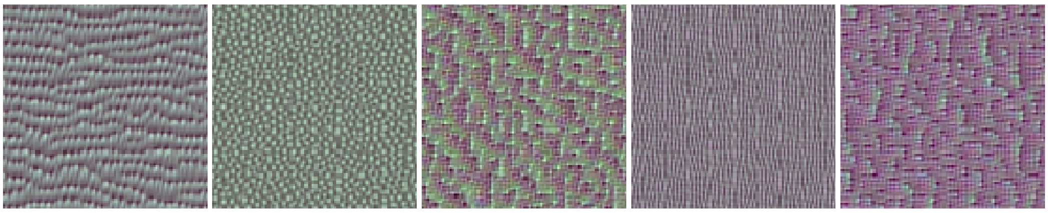
For gradient-based visualization methods, modification of the technique to calculate the gradient was implemented by registering a “new” ReLu function with a different approach to calculate the gradients. This new function then replaced the original ReLu function in a duplicated version of the model, which maintained the consistency of the original model.

The results of quality control of these gradient-based methods can be found in Figure 5.2, which shows gradient-based visualization of a kitten image by Springenberg *et al.* [20] (Figure 5.2b) and the *deoxys* framework (Figure 5.2c). While both models were trained on the ImageNet dataset, the architectures of the VGG16 model and the model used by Springenberg *et al.* [20] are different (see Tables B.1 and B.2 in Appendix B). This caused some discrepancy between the two results. Moreover, because the resulting images of these gradient-based methods contained pixels with the values outside the range of the Red-Green-Blue images (between 0 and 255), a normalization step (i.e., scaling the pixel values to be within the range of 0 and 255) had to apply to the three color channels. Since Springenberg *et al.* [20] did not describe the normalization function used in their results, the normalization step used in the *deoxys* framework may not be the same. This different step also contributed to the slight variations in results.

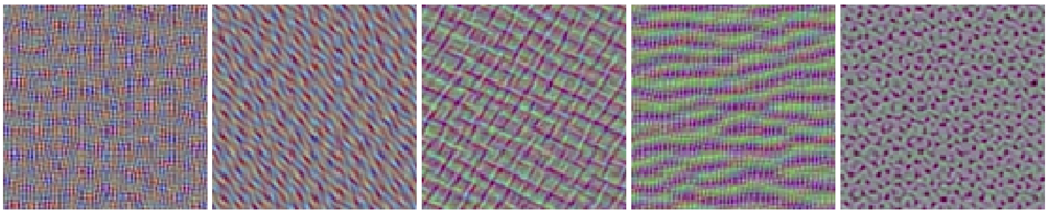
However, the similarity in the results between the two models was sufficient to let these implementations pass the quality control test. The similarity can be



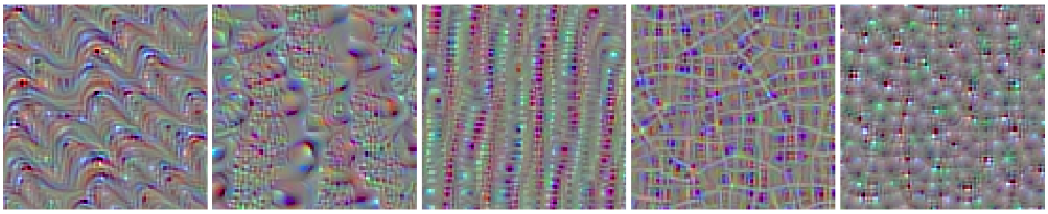
(a) block1_conv1



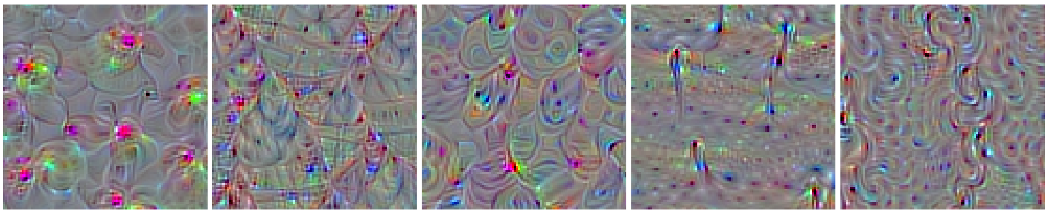
(b) block2_conv1



(c) block3_conv1

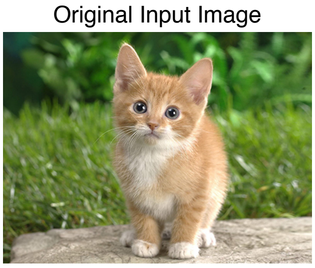


(d) block4_conv1

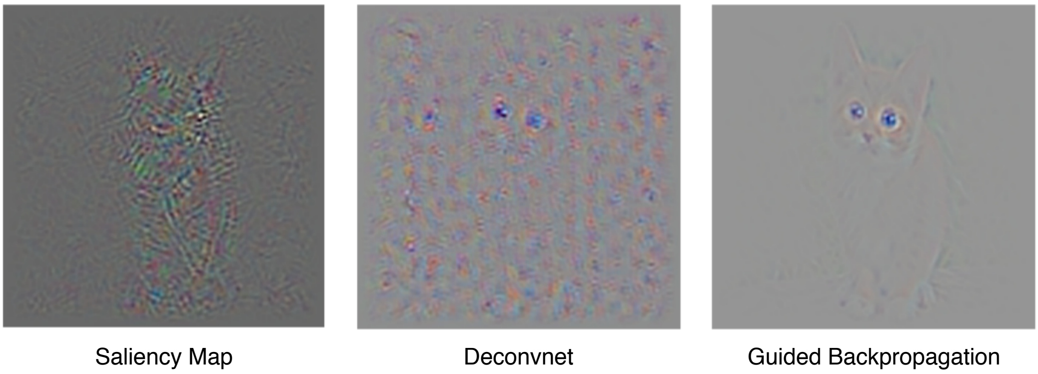


(e) block5_conv1

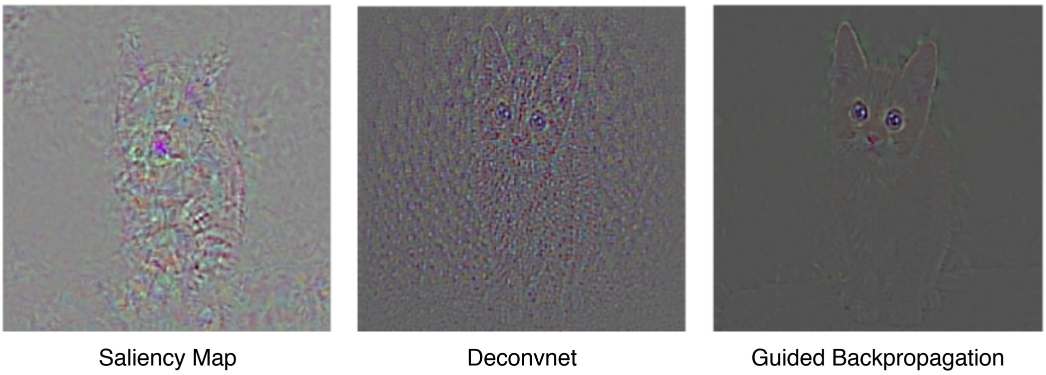
Figure 5.1: Illustration of VGG16 activation maximization results. Example results of activation maximization generated by the *deoxys* framework for five layers (a) to (e) of the VGG16 model trained on the ImageNet dataset (Appendix B). The final results were generated after 50 iterations in activation maximization.



(a) Original Image



(b) Results from Springenberg *et al.* [20].



(c) Results from *deoxys* framework.

Figure 5.2: Quality control results on saliency maps, deconvnet and guided backpropagation. From the original images (a) the saliency maps, deconvnet and guided backpropagation are generated by Springenberg *et al.* [20] (b) and the *deoxys* framework (c).

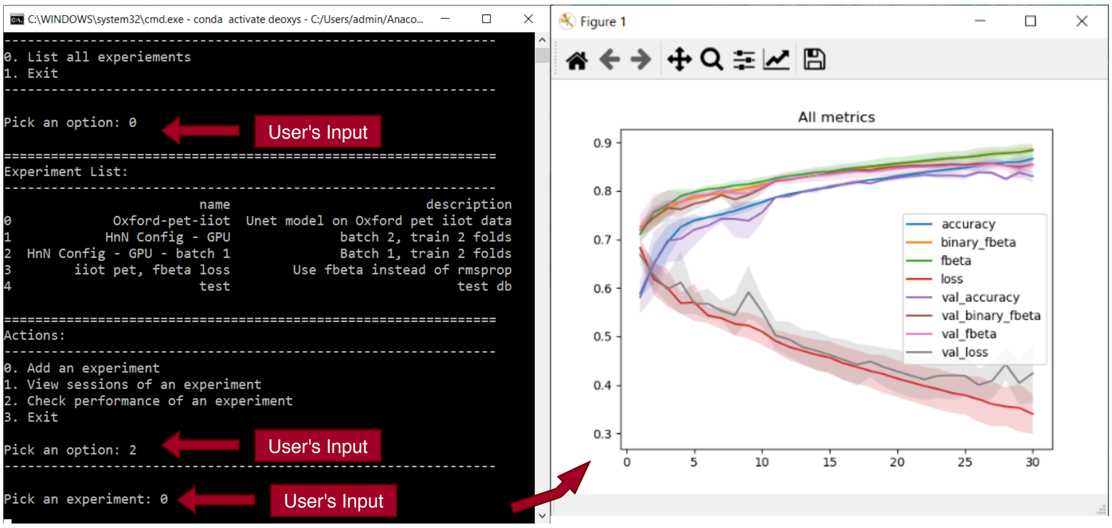
seen in the similar locations of important pixels in the saliency maps, together with the eyes and the noisiness in the deconvnet results, as well as the smooth and sharp highlights of the cat's eyes, ears and nose, and its edges in the guided backpropagation results. Generating these images was also easy using the APIs proposed in Section 3.2.1 on page 34 (see below lines 8 to 24).

```
1  # After importing all necessary libraries and
2  # load the vgg16 into deoxys model
3
4  # deo: the Deoxys Model containing the pre-train VGG16 model
5  # imgs: list of images for generating saliency maps,
6  # deconvnet and guided backpropagation
7
8  saliency_maps = deo.backprop(
9      'block5_conv3', # Layer's name
10     imgs, # List of images for calculation
11     mode='mean', # Mode for loss function
12     )
13
14 deconvnet = deo.deconv(
15     'block5_conv3', # Layer's name
16     imgs, # List of images for calculation
17     mode='mean', # Mode for loss function
18     )
19
20 guided_backpropagation = deo.guided_backprop(
21     'block5_conv3', # Layer's name
22     imgs, # List of images for calculation
23     mode='mean', # Mode for loss function
24     )
```

5.1.2 Database Integration

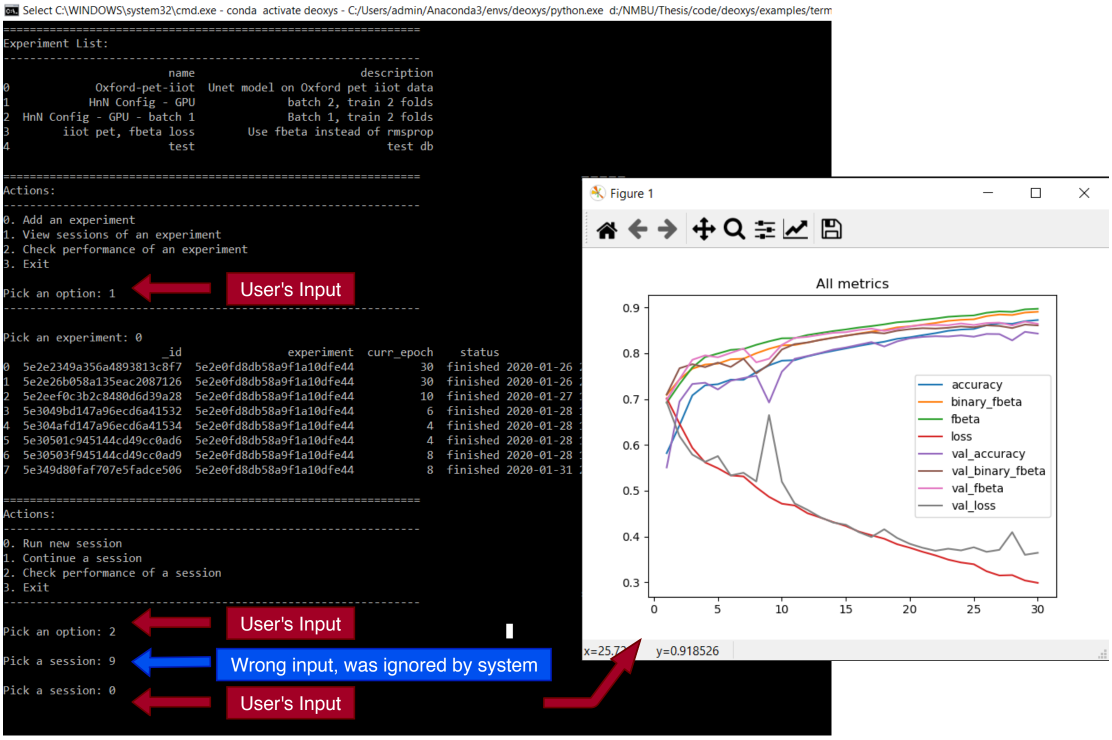
To manage training experiments, a database management system (DBMS) is essential. With database integration, managing the results of different runs from the same experiments is easier. This is because results from different experiments, as well as information about the locations of log-files, predictions, and saved models, are centralized and accessible. The performance analysis can be done directly thanks to the database, without manually putting everything into the same Excel file.

Manual tests were performed to ensure the reliability of database integration. The results of these tests are available in the full test report in Appendix C on page 123.



```
--------------------------------------------------------------
0. List all experiments
1. Exit
--------------------------------------------------------------

Pick an option: 0

==============================================================
Experiment List:
--------------------------------------------------------------
                         name                      description
0             Oxford-pet-iiot  Unet model on Oxford pet iiot data
1              HnN Config - GPU         batch 2, train 2 folds
2   HnN Config - GPU - batch 1          Batch 1, train 2 folds
3           iiot pet, fbeta loss    Use fbeta instead of rmsprop
4                         test                         test db

==============================================================
Actions:
--------------------------------------------------------------
0. Add an experiment
1. View sessions of an experiment
2. Check performance of an experiment
3. Exit

Pick an option: 2
--------------------------------------------------------------

Pick an experiment: 0
```

(a) The average performance of an experiment.


Select C:\WINDOWS\system32\cmd.exe - conda activate deoxys - C:/Users/admin/Anaconda3/envs/deoxys/python.exe d:/NMBU/Thesis/code/deoxys/examples/term
User's Input
User's Input
Wrong input, was ignored by system
User's Input
Figure 1
All metrics
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
30
accuracy
binary_fbeta
fbeta
loss
val_accuracy
val_binary_fbeta
val_fbeta
val_loss
y=0.918526

```
==================================================================
Experiment List:
------------------------------------------------------------------
                         name                      description
0             Oxford-pet-iiot  Unet model on Oxford pet iiot data
1              HnN Config - GPU         batch 2, train 2 folds
2   HnN Config - GPU - batch 1          Batch 1, train 2 folds
3           iiot pet, fbeta loss    Use fbeta instead of rmsprop
4                         test                         test db

==================================================================
Actions:
------------------------------------------------------------------
0. Add an experiment
1. View sessions of an experiment
2. Check performance of an experiment
3. Exit

Pick an option: 1
------------------------------------------------------------------

Pick an experiment: 0
                        _id                experiment  curr_epoch   status
0  5e2e2349a356a4893813c8f7  5e2e0fd8db58a9f1a10dfe44          30  finished 2020-01-26
1  5e2e26b058a135eac2087126  5e2e0fd8db58a9f1a10dfe44          30  finished 2020-01-26
2  5e2eef0c3b2c8480d6d39a28  5e2e0fd8db58a9f1a10dfe44          10  finished 2020-01-27
3  5e3049bd147a96ecd6a41532  5e2e0fd8db58a9f1a10dfe44           6  finished 2020-01-28
4  5e304afd147a96ecd6a41534  5e2e0fd8db58a9f1a10dfe44           4  finished 2020-01-28
5  5e30501c945144cd49cc0ad6  5e2e0fd8db58a9f1a10dfe44           4  finished 2020-01-28
6  5e30503f945144cd49cc0ad9  5e2e0fd8db58a9f1a10dfe44           8  finished 2020-01-28
7  5e349d80faf707e5fadce506  5e2e0fd8db58a9f1a10dfe44           8  finished 2020-01-31

==================================================================
Actions:
------------------------------------------------------------------
0. Run new session
1. Continue a session
2. Check performance of a session
3. Exit
------------------------------------------------------------------

Pick an option: 2

Pick a session: 9

Pick a session: 0
```

(b) The performance of a session of an experiment.

Figure 5.3: Screenshots from the example terminal application integrated with database. In these screenshots, users can choose from lists of actions to interact with the *deoxys* framework, including viewing the average performance of an experiment (a) and viewing the performance of a single session of an experiment (b).

From the report, all defined test cases satisfied the expected results when the *deoxys* framework interacted with the DBMS.

An example usage of the database integration is available at `https://github.com/huynhngoc/deoxys/blob/master/examples/terminal-app-hn-example.py`. This example is a terminal application derived from the *deoxys* framework, where users can interactively (1) configure a new experiment, (2) run one or more experiments, (3) continue one or more experiments, (4) view the performance of one experiment, and (5) view the overall performance of a group of experiments. Screenshots of this terminal application can be found in Figure 5.3. In this figure, after choosing from lists of actions, a user can see the average performance of all runs of an experiment (Figure 5.3a), or just view one session of that experiment (Figure 5.3b). Other options to choose from this terminal application can be found in Figure 5.3.

5.2 Visualization Results

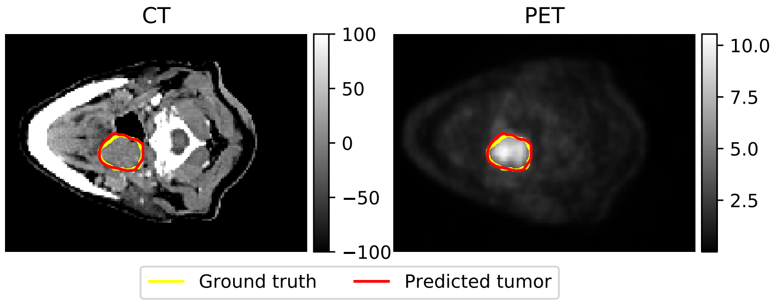
This section provides the results obtained from the visualization methods, as described in Chapter 4, applied to the Unet model (see Figure 4.1 on page 42) and trained on the head and neck cancer dataset consisting of PET/CT images.

5.2.1 Patterns extracted from the Unet model

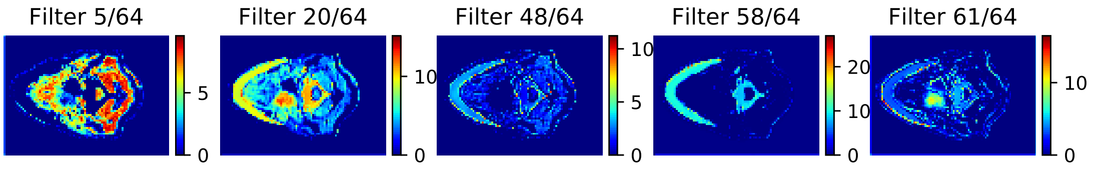
Activation Map

Activation maps for the head and neck cancer dataset generated according to Section 4.2.1 on page 45 were analyzed. Figures 5.4 to 5.9 show examples of patterns extracted by some filters for six patients (as listed in Table 4.2 on page 43). In addition, the full activation maps of all convolutional layers can be found in Appendix E on page 131.

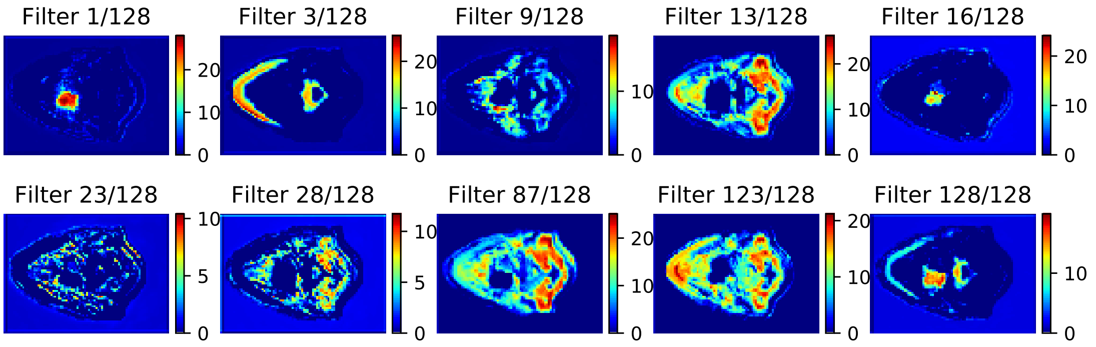
When looking at the activation maps, we can see that the first few layers extracted areas with similar intensity values, which formed into parts of the head (or neck). The tongues, bones (jaws and spine), muscles, glands, etc. from the CT channel, as well as the lymph nodes in the PET channel can be found separately or together in each filter in the activation maps of these layers. For example, when considering slice 86 of patient 91, at the conv2d_1 layer (Figure 5.4b), filter 5 extracted the capitis muscles and healthy parts of the sublingual and submandibular spaces (the



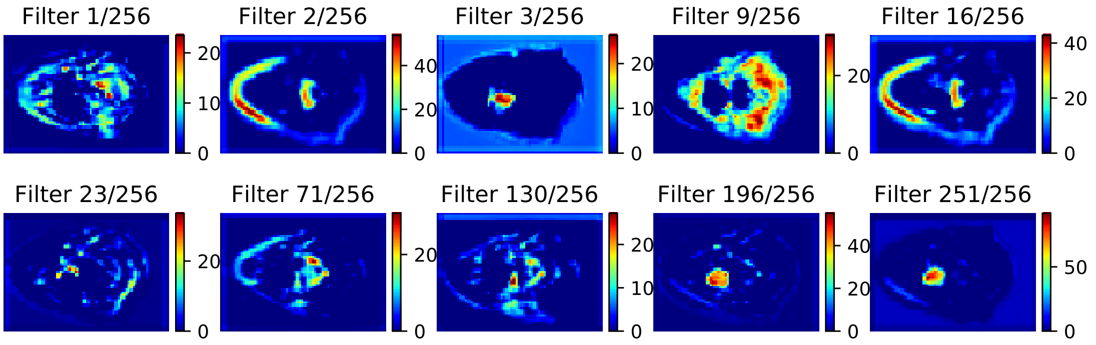
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

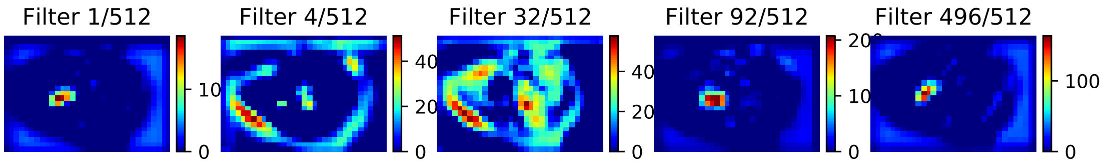


(c) Layer conv2d_3

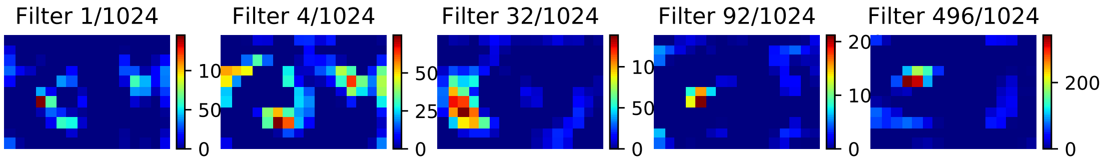


(d) Layer conv2d_5

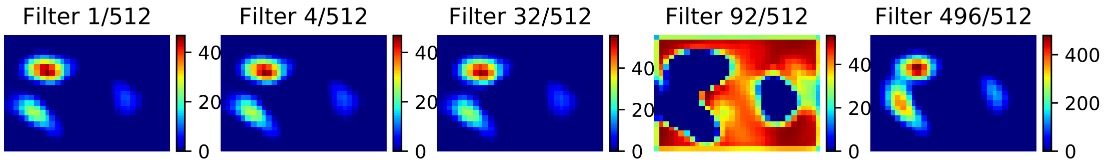
Figure 5.4: Activation Maps for patient 91, slice 86, with a Dice score of 0.94. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the Hounsfield Units (HU) [49] of the CT scan and the Standardized Uptake Value (SUV) [50] of the PET scan, in which high values result in whiter colors. In b-j, the color bars show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



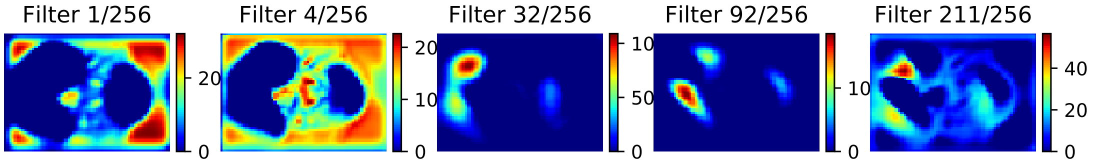
(e) Layer conv2d_7



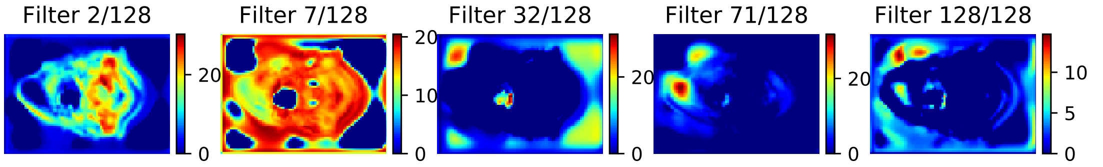
(f) Layer conv2d_9



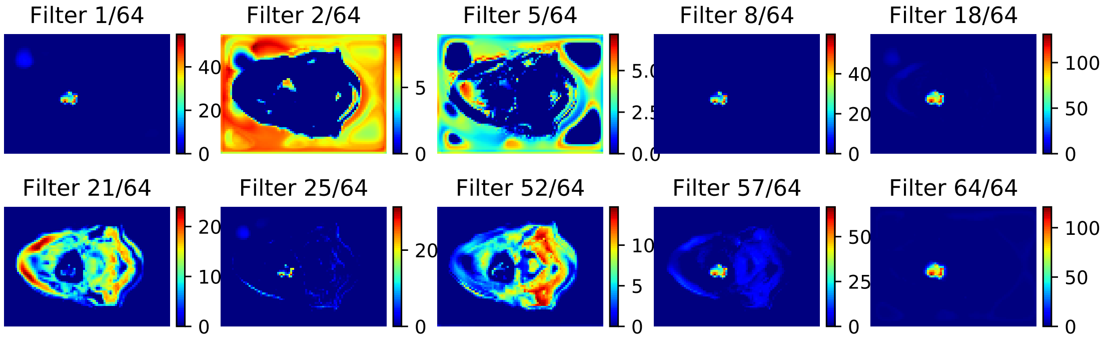
(g) Layer conv2d_11



(h) Layer conv2d_13

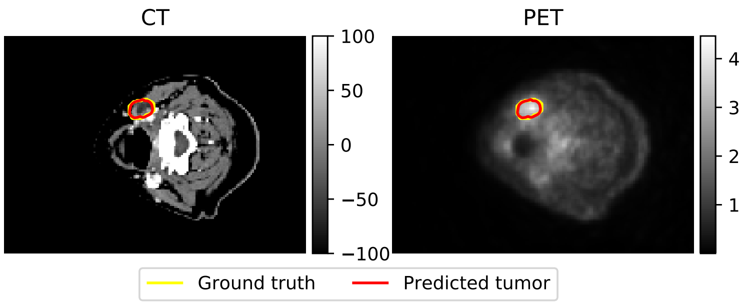


(i) Layer conv2d_15

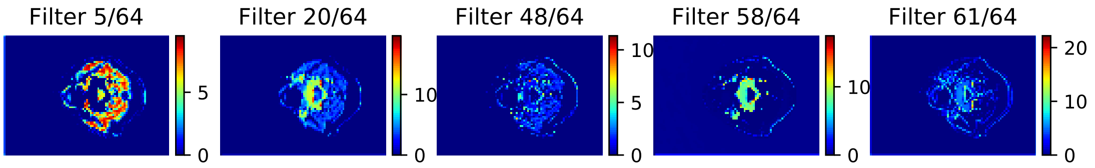


(j) Layer conv2d_17

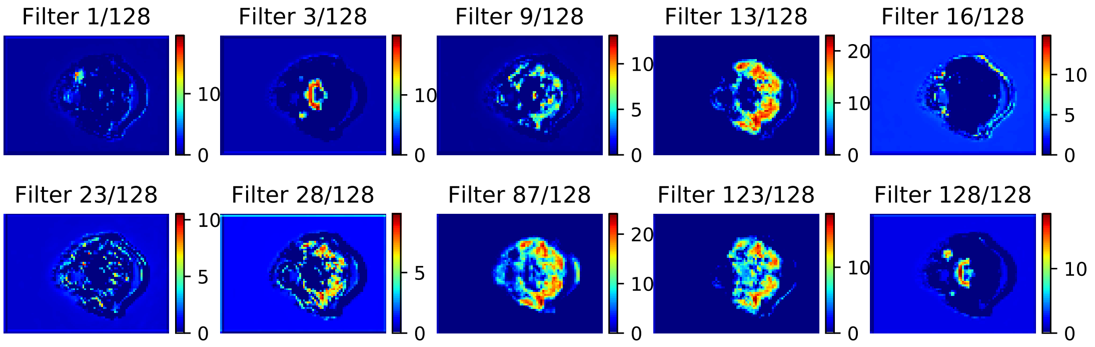
Figure 5.4: Continued.



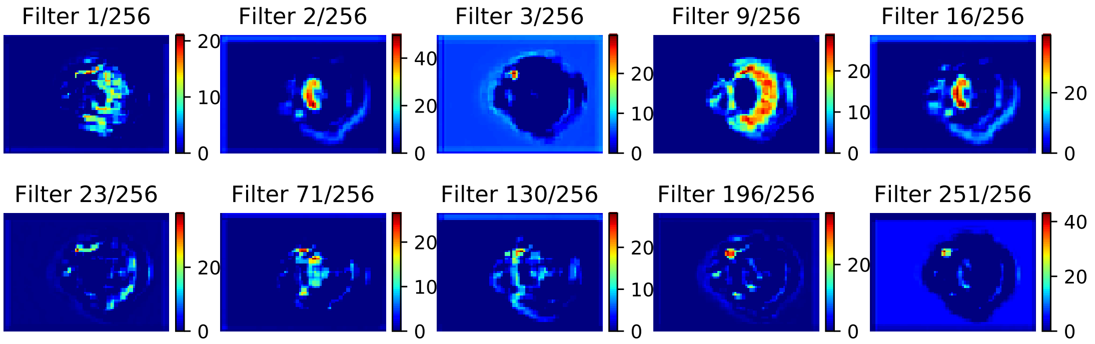
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

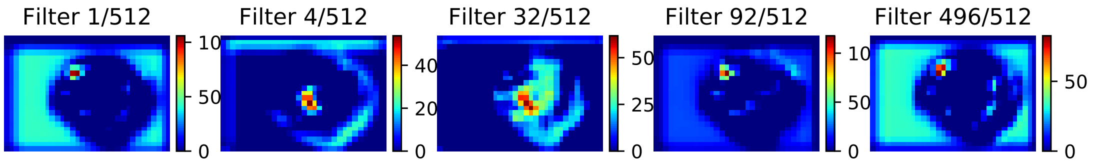


(c) Layer conv2d_3

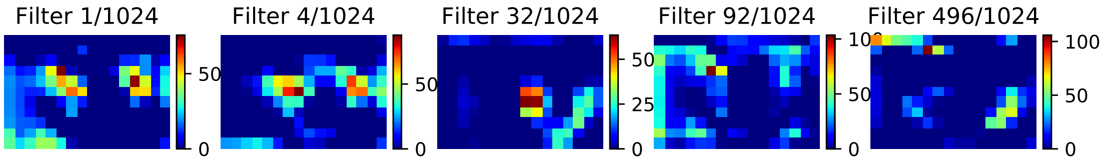


(d) Layer conv2d_5

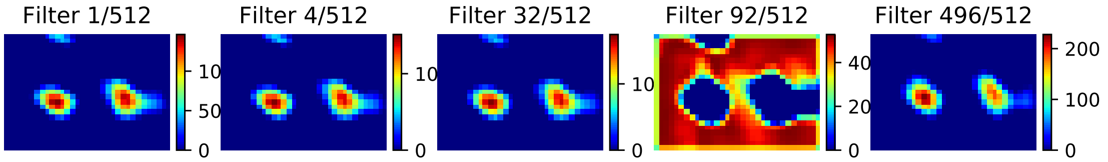
Figure 5.5: Activation Maps for patient 148, slice 11, with a Dice score of 0.94. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the HU of the CT scan and the SUV of the PET scan, in which higher values result in whiter colors. The color bars in b-j show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



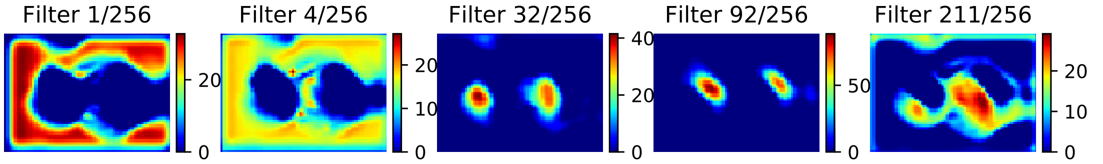
(e) Layer conv2d_7



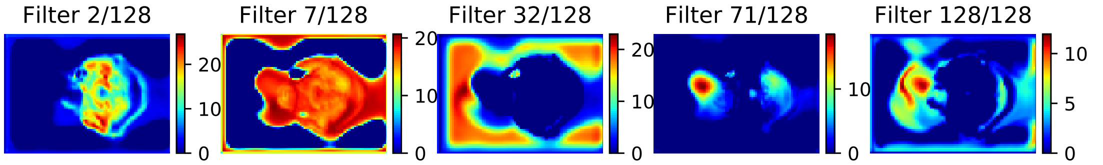
(f) Layer conv2d_9



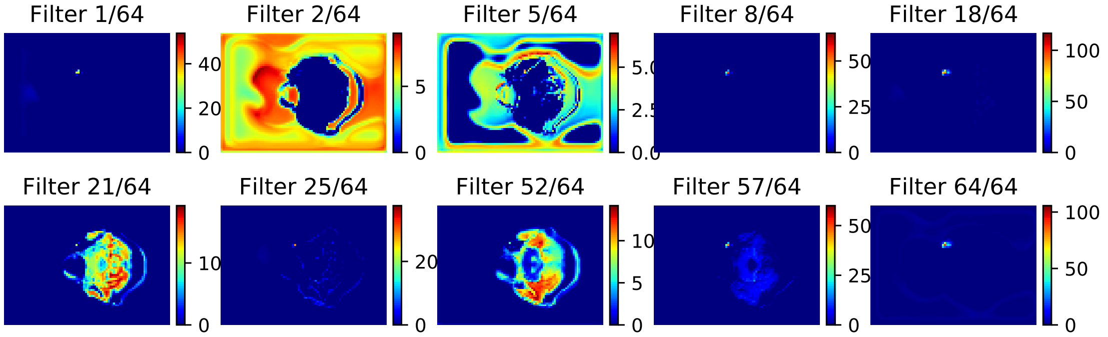
(g) Layer conv2d_11



(h) Layer conv2d_13

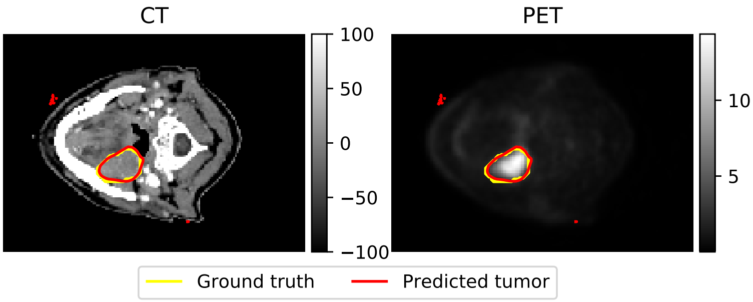


(i) Layer conv2d_15

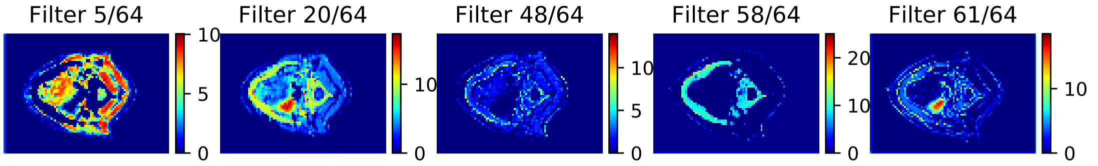


(j) Layer conv2d_17

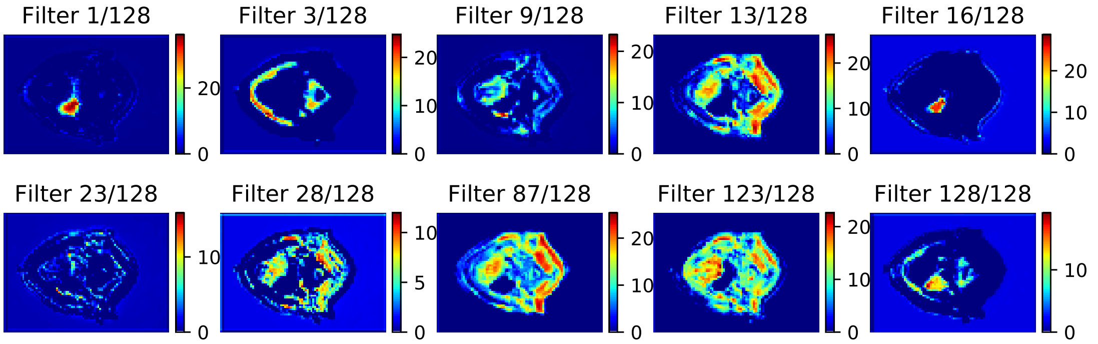
Figure 5.5: Continued.



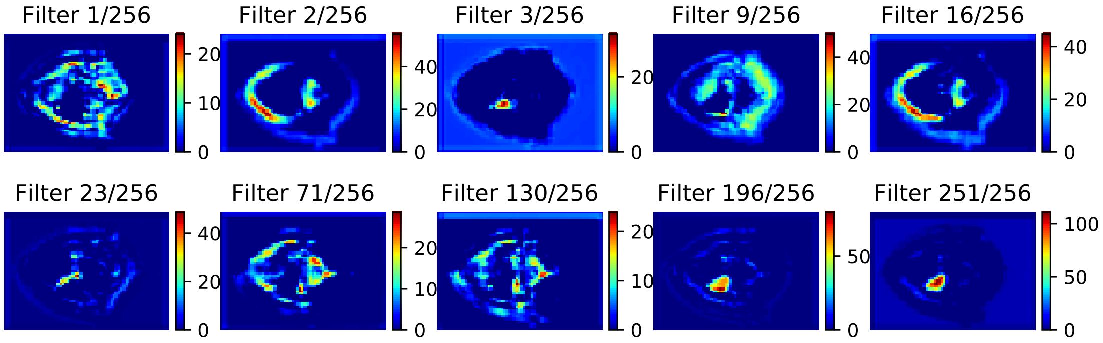
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

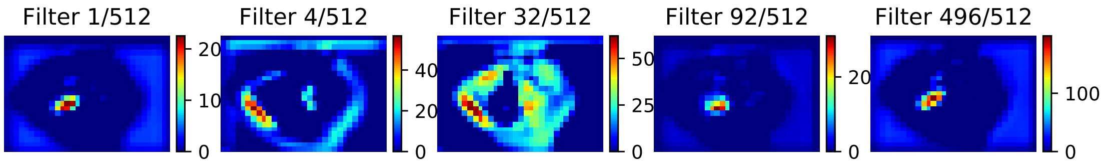


(c) Layer conv2d_3

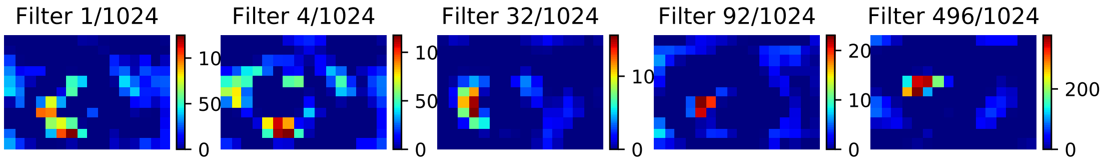


(d) Layer conv2d_5

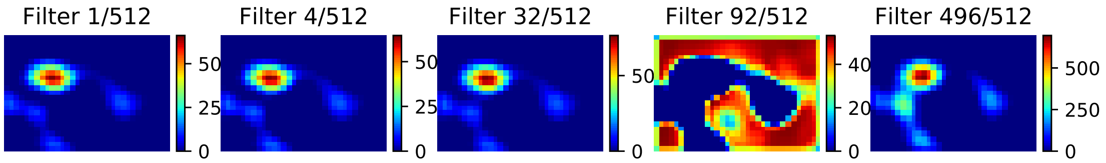
Figure 5.6: Activation Maps for patient 209, slice 14, with a Dice score of 0.94. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the HU of the CT scan and the SUV of the PET scan, in which higher values result in whiter colors. The color bars in b-j show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



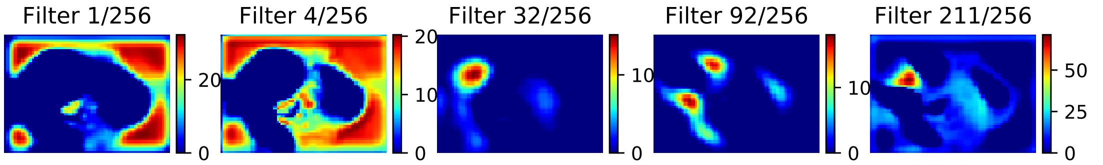
(e) Layer conv2d_7



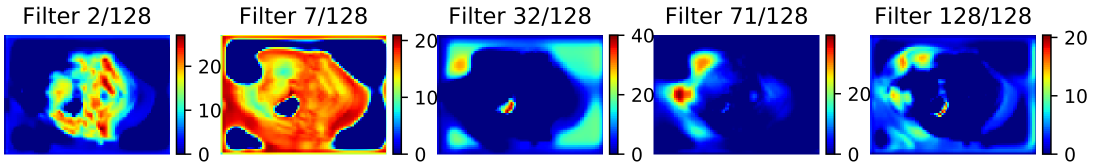
(f) Layer conv2d_9



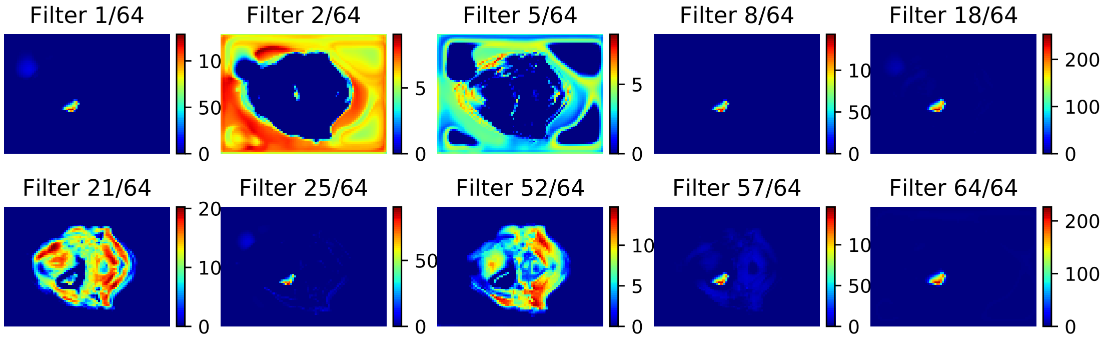
(g) Layer conv2d_11



(h) Layer conv2d_13

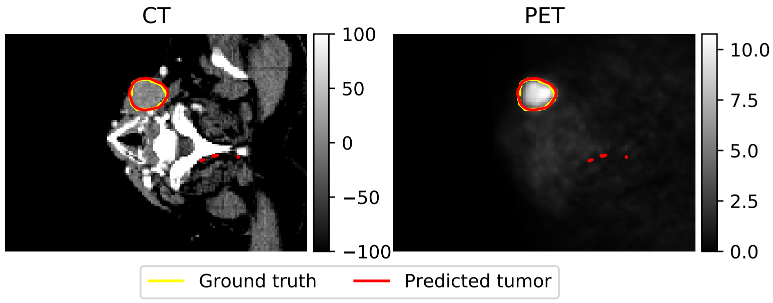


(i) Layer conv2d_15

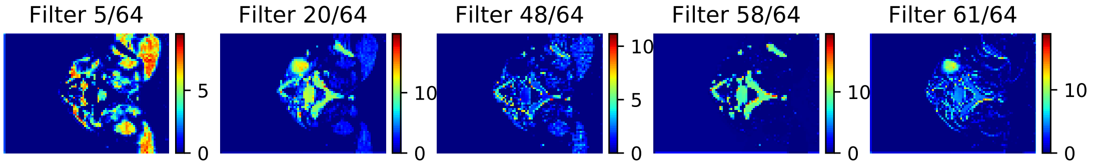


(j) Layer conv2d_17

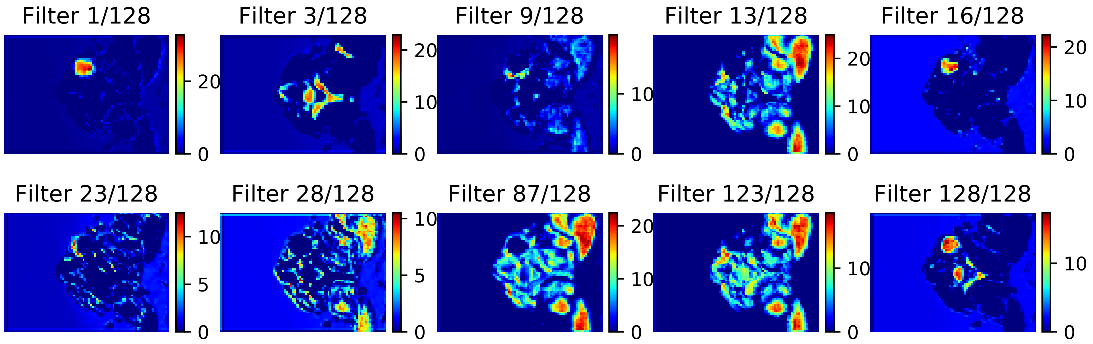
Figure 5.6: Continued.



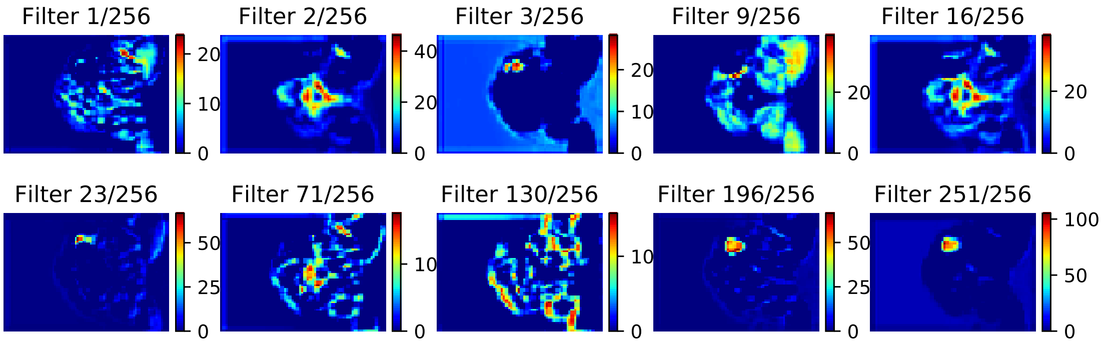
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

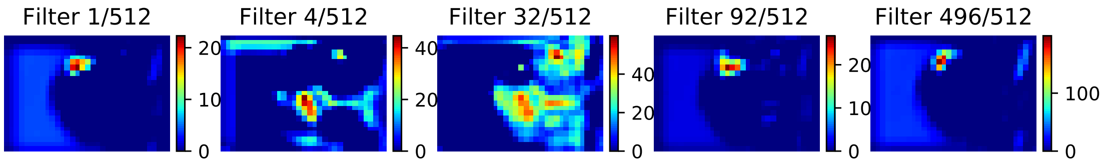


(c) Layer conv2d_3

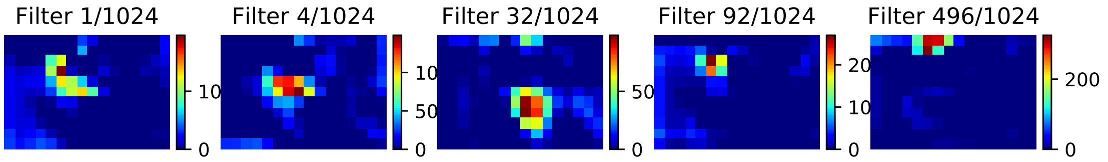


(d) Layer conv2d_5

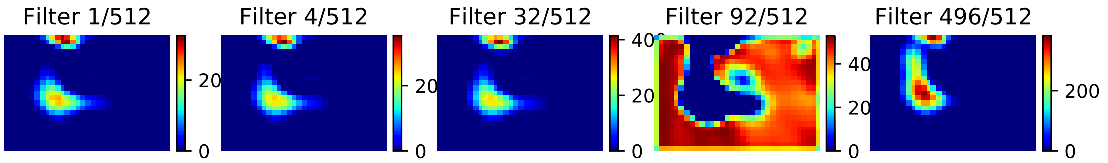
Figure 5.7: Activation Maps for patient 217, slice 20, with a Dice score of 0.95. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the HU of the CT scan and the SUV of the PET scan, in which higher values result in whiter colors. The color bars in b-j show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



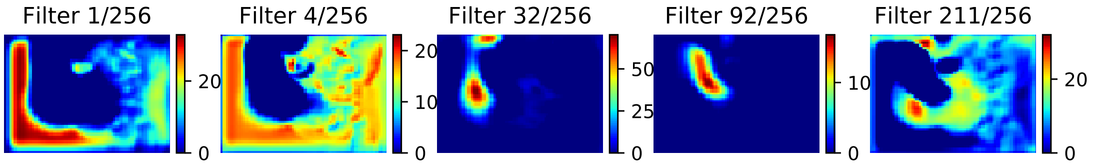
(e) Layer conv2d_7



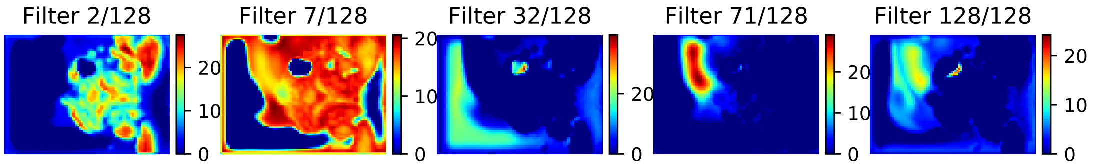
(f) Layer conv2d_9



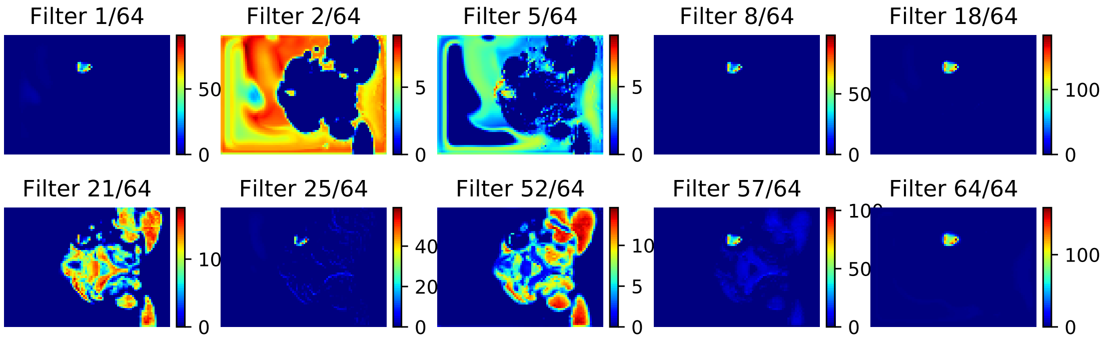
(g) Layer conv2d_11



(h) Layer conv2d_13

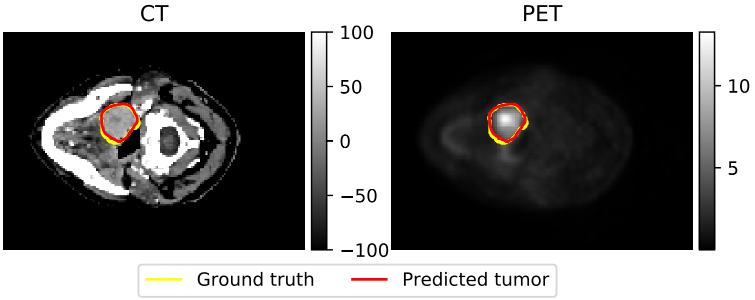


(i) Layer conv2d_15

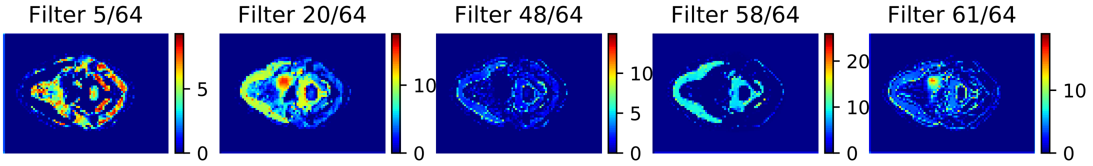


(j) Layer conv2d_17

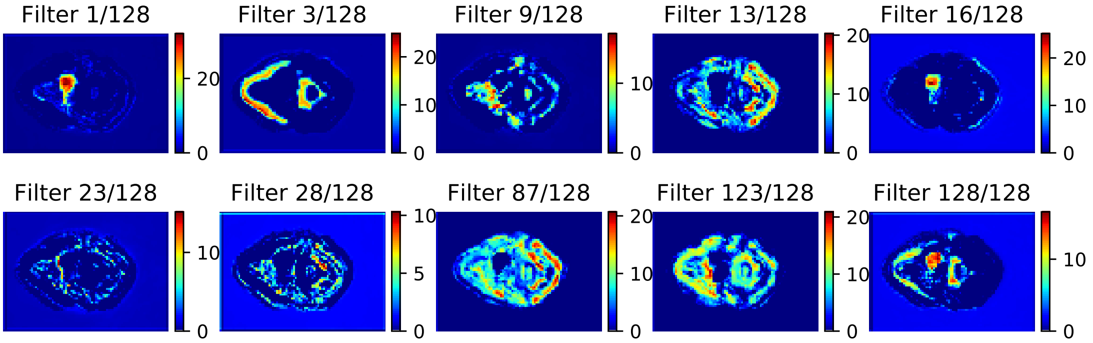
Figure 5.7: Continued.



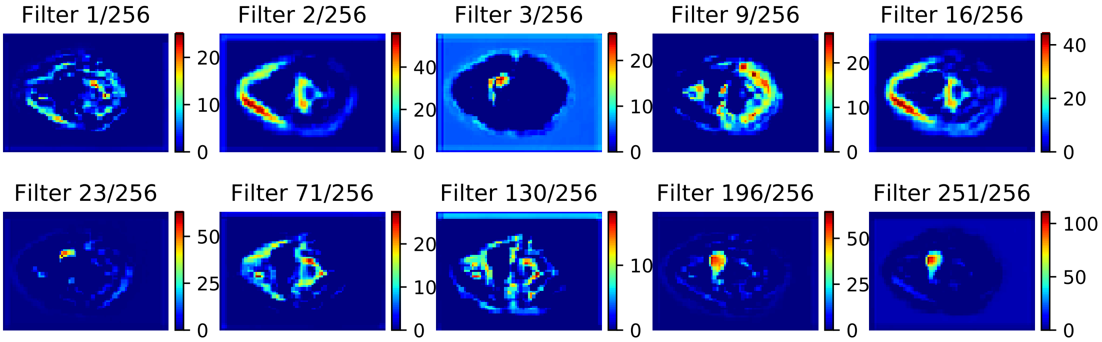
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

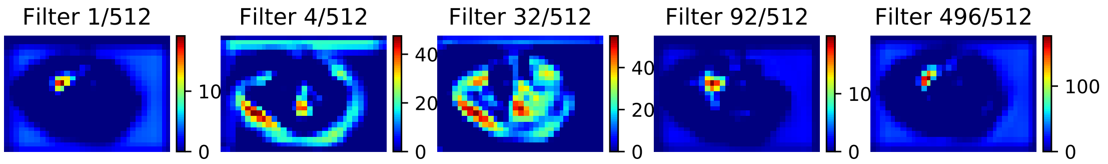


(c) Layer conv2d_3

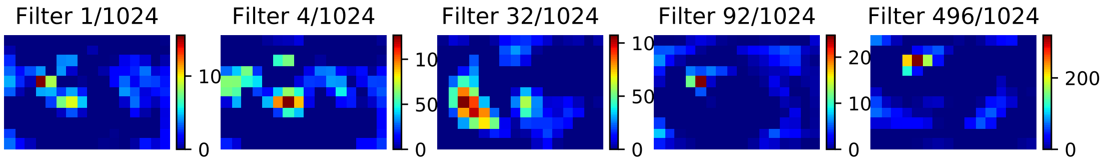


(d) Layer conv2d_5

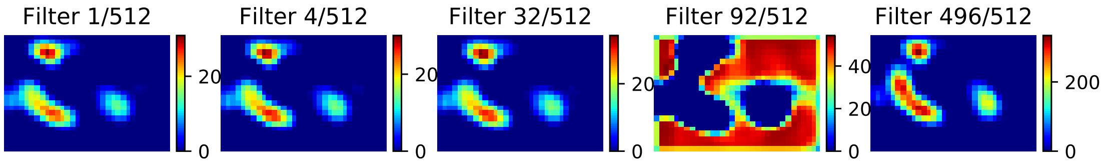
Figure 5.8: Activation Maps for patient 233, slice 48, with a Dice score of 0.95. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the HU of the CT scan and the SUV of the PET scan, in which higher values result in whiter colors. The color bars in b-j show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



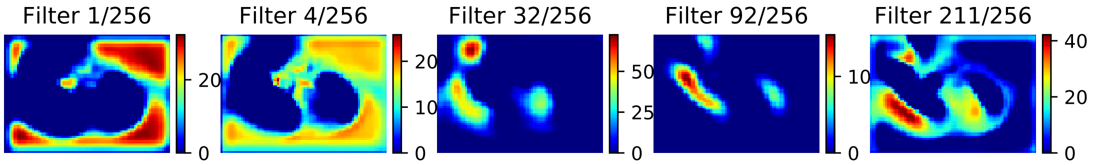
(e) Layer conv2d_7



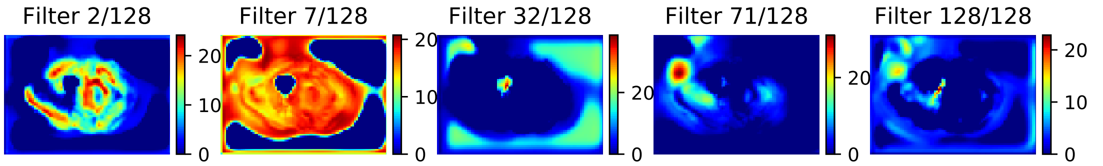
(f) Layer conv2d_9



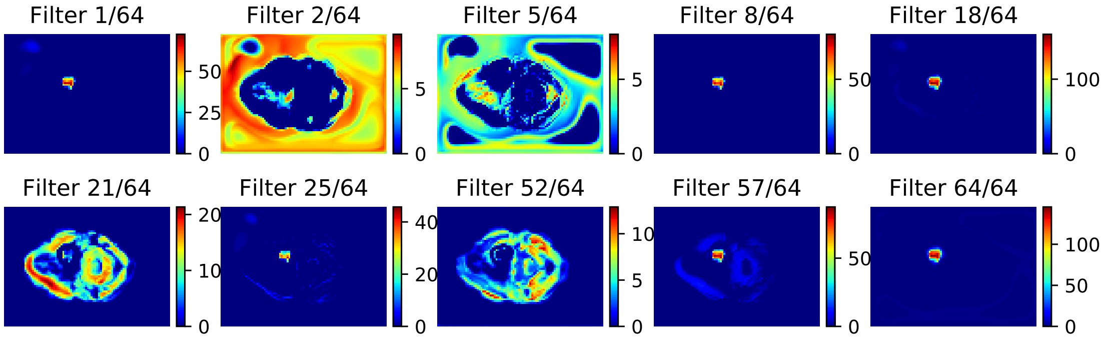
(g) Layer conv2d_11



(h) Layer conv2d_13

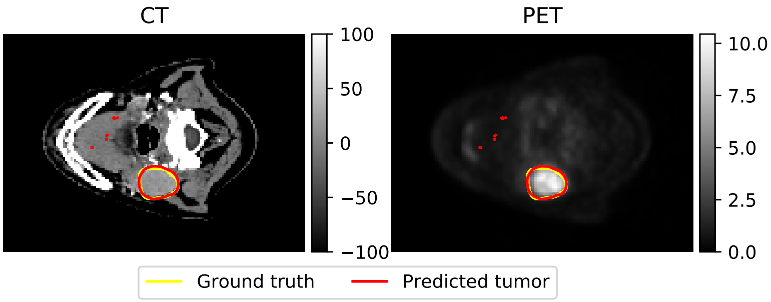


(i) Layer conv2d_15

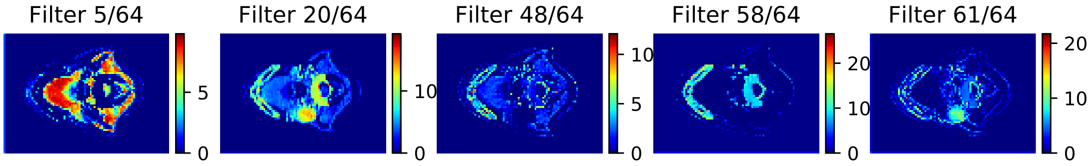


(j) Layer conv2d_17

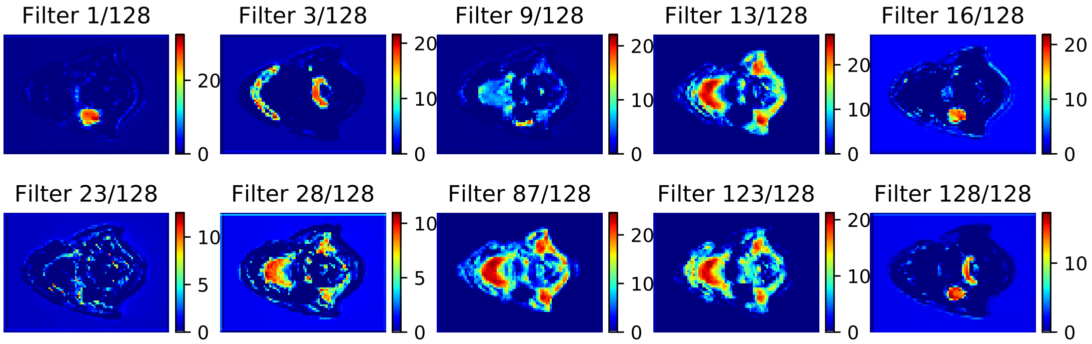
Figure 5.8: Continued.



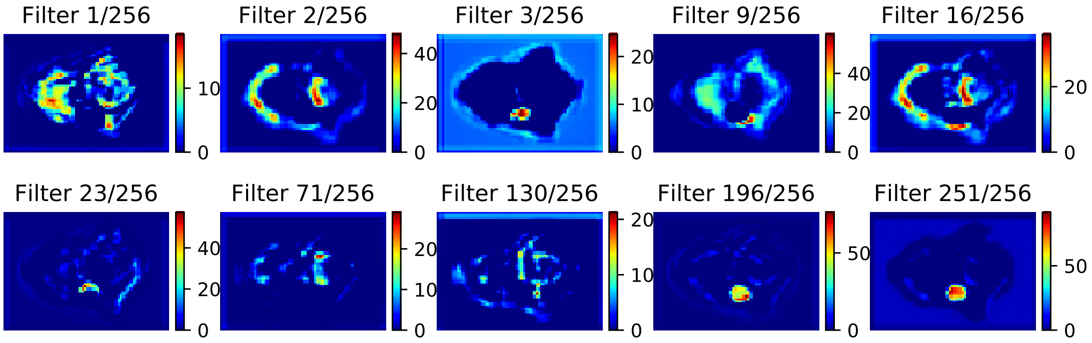
(a) Original image with two channels, CT and PET



(b) Layer conv2d_1

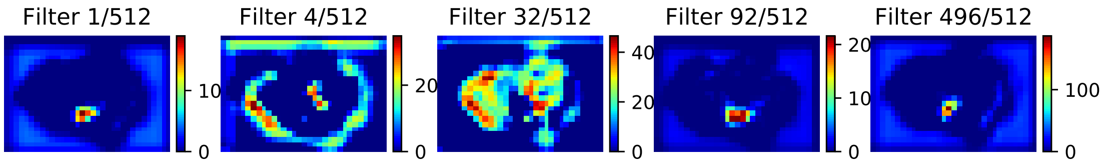


(c) Layer conv2d_3

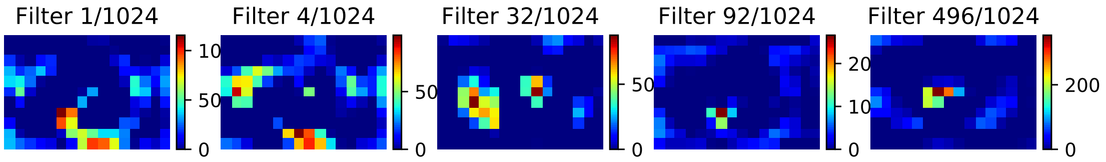


(d) Layer conv2d_5

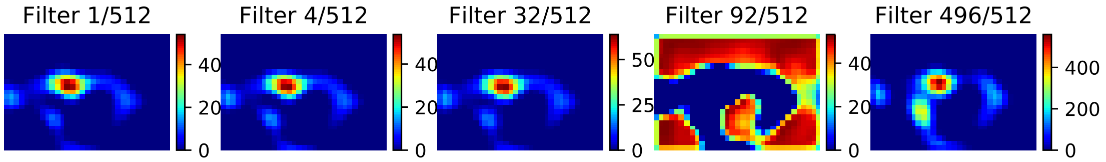
Figure 5.9: Activation Maps for patient 249, slice 55, with a Dice score of 0.95. From the original images (a) with CT and PET channel, each filter in each layer of the CNN model extracted different parts of the image. Examples of patterns extracted by some filters are showed in b-j. Layers in b to d are in the down-sampling path; layers e to g are layers at the bottom of the Unet; and layers h to j are in the up-sampling path. The color bars in (a) show the HU of the CT scan and the SUV of the PET scan, in which higher values result in whiter colors. The color bars in b-j show the intensity values of images in activation maps, where higher values result in more reddish colors. (*Continued on next page.*)



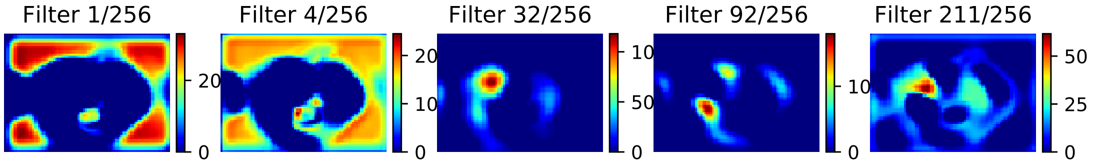
(e) Layer conv2d_7



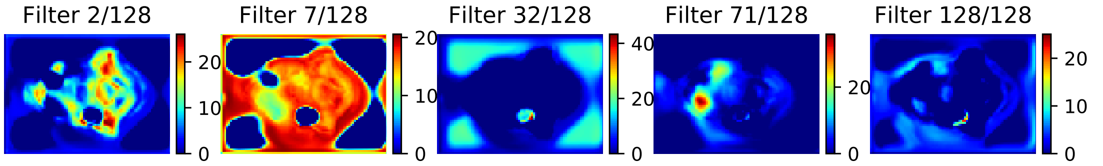
(f) Layer conv2d_9



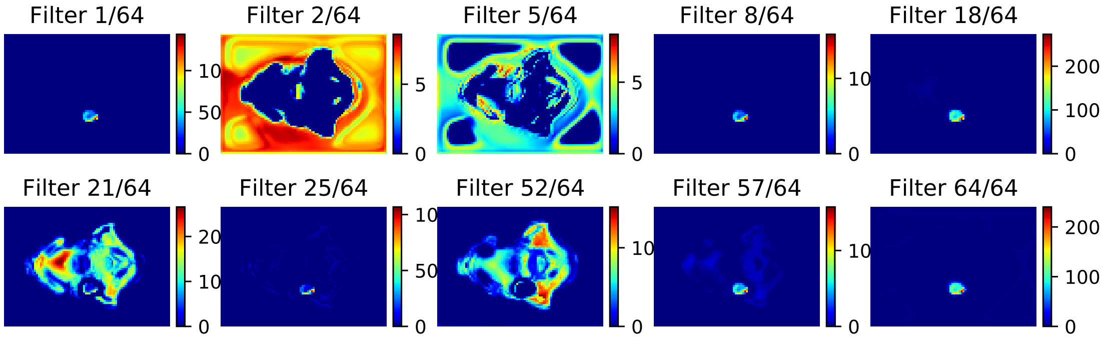
(g) Layer conv2d_11



(h) Layer conv2d_13



(i) Layer conv2d_15



(j) Layer conv2d_17

Figure 5.9: Continued.

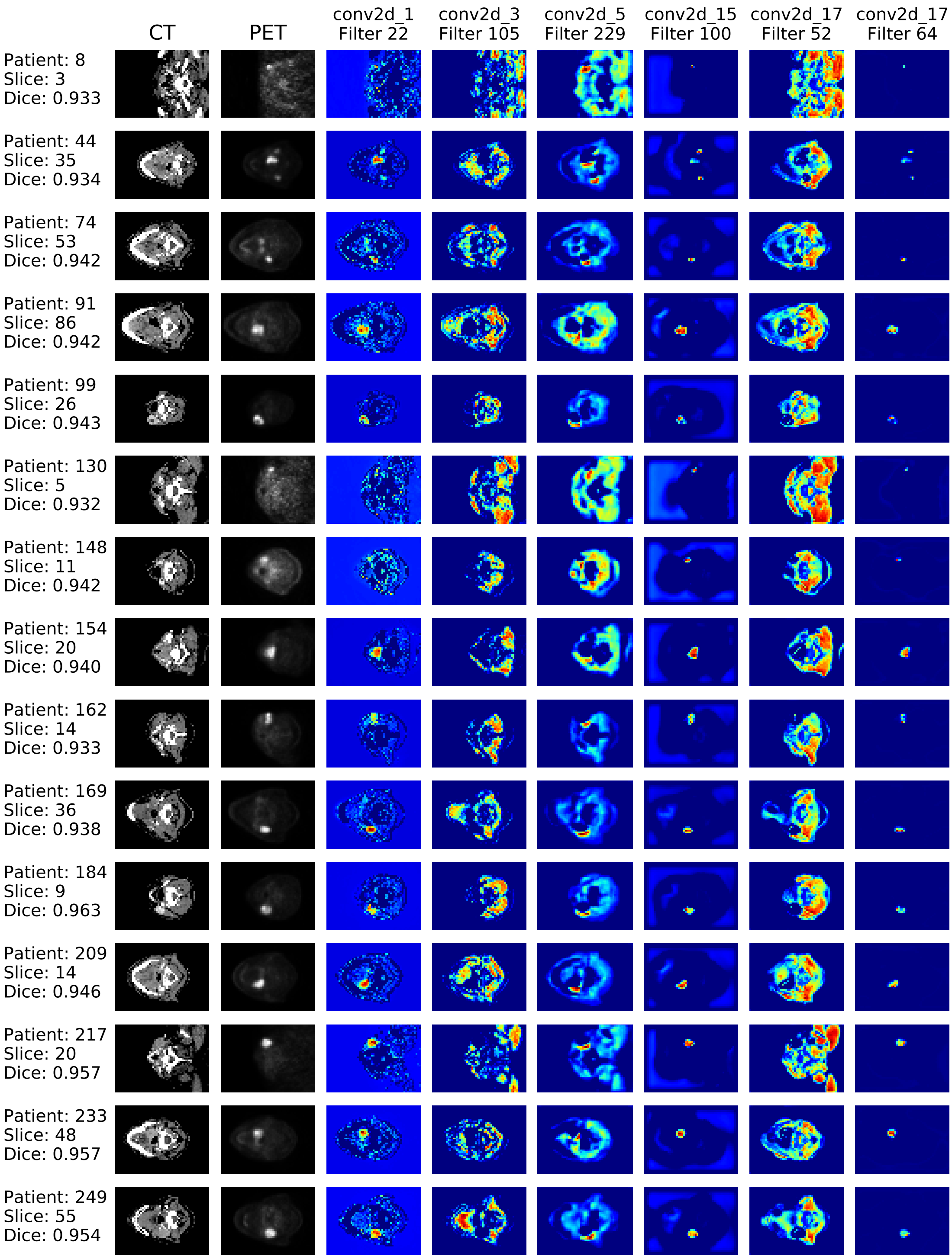

CT
PET
conv2d_1 Filter 22
conv2d_3 Filter 105
conv2d_5 Filter 229
conv2d_15 Filter 100
conv2d_17 Filter 52
conv2d_17 Filter 64
Patient: 8 Slice: 3 Dice: 0.933
Patient: 44 Slice: 35 Dice: 0.934
Patient: 74 Slice: 53 Dice: 0.942
Patient: 91 Slice: 86 Dice: 0.942
Patient: 99 Slice: 26 Dice: 0.943
Patient: 130 Slice: 5 Dice: 0.932
Patient: 148 Slice: 11 Dice: 0.942
Patient: 154 Slice: 20 Dice: 0.940
Patient: 162 Slice: 14 Dice: 0.933
Patient: 169 Slice: 36 Dice: 0.938
Patient: 184 Slice: 9 Dice: 0.963
Patient: 209 Slice: 14 Dice: 0.946
Patient: 217 Slice: 20 Dice: 0.957
Patient: 233 Slice: 48 Dice: 0.957
Patient: 249 Slice: 55 Dice: 0.954

Figure 5.10: Features extracted from different input images using the same filter. The first column contains the patient index and slice index of the input images, together with the Dice scores of the segmentation results. The CT and PET channels of the input images are in the next two columns. The remaining columns show the features extracted from these images, at specific layers of the Unet model Figure 4.1.

tongues together with the muscles and glands around it), while filter 20 extracted the bones from the CT channel and the lymph nodes from the PET channel. Filter 48 extracted most parts from the CT channel while filter 58 extracted only the bones. Filter 61 was similar to filter 20, except that the lymph nodes were more noticeable. In the next few layers, for example the conv2d_3 and conv2d_5 layers (Figure 5.4c and d), the features extracted from conv2d_1 layer were continued to be extracted, with smaller details or with some combinations. Nevertheless, the extracted features are still human readable.

Similar results can be found in slice 14 of patient 209 (Figure 5.6b to d) and slice 48 of patient 233 (Figure 5.8b to d), where the CT channel had similar structures while the lymph nodes' locations were in different locations in the sublingual and submandibular spaces. We can also see the full sublingual and submandibular spaces extracted from filter 5 of conv2d_1 layer in the case of patient 249, slice 55 (Figure 5.9b) as the lymph nodes were outside those spaces. For slices at different locations, where the jaws and tongues were not visible, in the case of PET/CT scans at the neck (slice 11 of patient 148 in Figure 5.5 and slice 20 of patient 217 in Figure 5.7), these non-existent parts were not included in the extracted features, but similar areas such as bones and lymph nodes were still extracted using the same filters.

Features extracted from layers at the bottom of the Unet architecture (Figure 4.1) were smaller and more abstract (Figures (e) to (g) of Figures 5.4 to 5.9), which made it difficult to make any conclusions. For layers in the up-sampling path, the features extracted were either similar to the segmentation results, or contained some information from the original input images (Figures (h) to (j) of Figures 5.4 to 5.9).

The same filter at a specific layer extracted the same information from different input images (see Figure 5.10). These features included parts with low intensity values in the original images, which were enhanced significantly using the activation maps. Therefore, if we know the layers' names and filters' indices that extract some specific parts or patterns of the input images, we can look at that filter directly using the activation maps when we want to check on those features.

In the same layer of the activation maps, many filters extracted similar features. Although the outputs of these filters were not exactly the same, they provided the same information, as similar features were "highlighted", but with different levels of intensity. For example, over 20 out of 127 filters in conv2d_3 layer extracted the jaws and spiral bone in the CT channel, and over 25 out of 127 filter in that same layer extracted the lymph nodes in the PET channel. This may be because

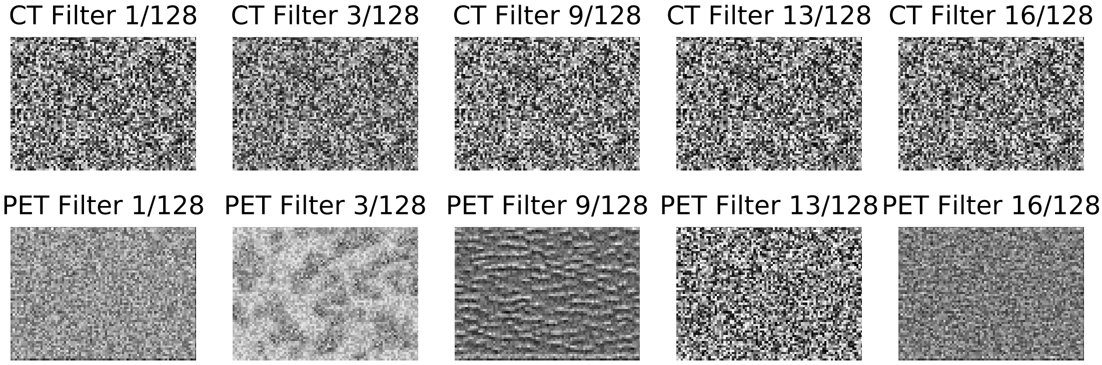
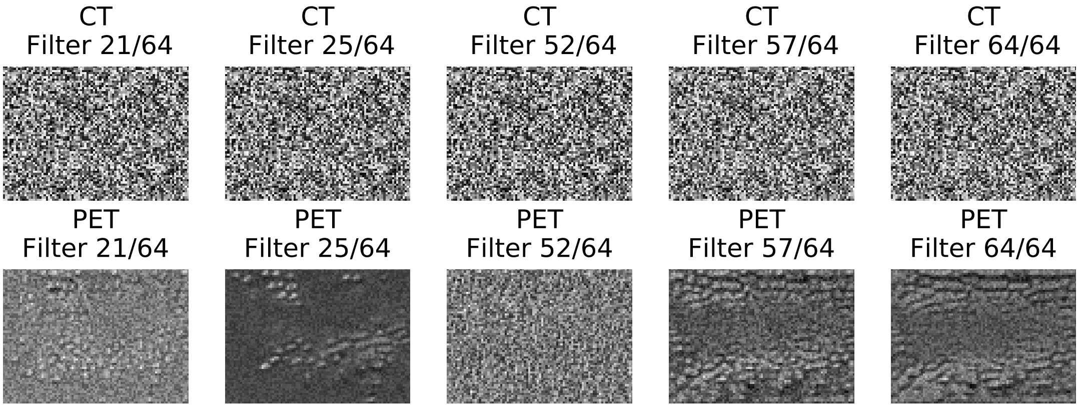


Figure 5.11: Activation maximization results at filters in the conv2d_3 layer, which belongs to the downsampling path of the Unet model. The resulting images consisted of the CT channel and the PET channel, which maximized filters in the conv2d_3 layer after applying 20 iterations of activation maximization to the initial images with random noise (see Figure 2.8 for details).

the model was too complex, which caused redundancy in some of the filters, or the model did not learn enough to utilize all of its filters.

Activation Maximization

To check the patterns and features that filters in the convolutional layers searched for, we can use the results from activation maximization. However, the images generated by using activation maximization methods according to Section 4.2.2 on page 45 did not provide enough information. Most of the generated images were noisy and abstract, as the initial image with random noise did not transform in the activation maximization process (Figure 2.8 on page 22). Only a few images generated from some specific filters contained the PET channels with blob-like patterns, while the CT channel remained noisy. The example results after 20 iterations can be found in Figures 5.11 and 5.12. The results after continuing the activation maximization process with more iterations remained unchanged.



CT Filter 21/64, CT Filter 25/64, CT Filter 52/64, CT Filter 57/64, CT Filter 64/64

PET Filter 21/64, PET Filter 25/64, PET Filter 52/64, PET Filter 57/64, PET Filter 64/64

Figure 5.12: Activation maximization results at filters in the conv2d_17 layer, which belongs to the upsampling path of the Unet model. The resulting images consisted of the CT channel and the PET channel, which maximized filters in the conv2d_17 layer after applying 20 iterations of activation maximization to the initial images with random noise (see Figure 2.8 for details).

5.2.2 Pixel contributions to the delineation

This section provides the results of pixel importance generated using the gradient-based visualization methods (the saliency map, the deconvnet and guided backpropagation methods) according to Section 4.2.3 on page 45. In Figures 5.13 to 5.23, the importance of pixels in each image channel (PET or CT) were plotted separately. Since the resulting images were the gradients of the calculated loss score, which was either the result of the *positive prediction* or *true positive* loss function, with respect to the original images, three aspects of the gradients were considered: the positive gradients (positively correlated with the loss score), the negative gradients (negatively correlated with the loss score) and the total magnitude of the gradients (both positive and negative gradients). Pixels with zero gradients had no impacts on the loss score.

Images with high Dice scores

For images with high Dice scores, the gradient-based visualization results of the same image using the two different loss functions (*positive prediction* and *true positive*) were similar. This is explainable as these loss functions will yield similar results when the output tensor $\hat{Y}$ and the ground truth tensor Y are similar (see the below equations).

Given that

$$J_{positive\_prediction}(\hat{Y}) = \sum (\hat{y}_{ij} \cdot f_{threshold}(\hat{y}_{ij}))$$
$$J_{true\_positive}(\hat{Y}) = \sum (\hat{y}_{ij} \cdot f_{threshold}(y_{ij}))$$

When $\hat{Y} \approx Y$, $J_{positive\_prediction}(\hat{Y}) \approx J_{true\_positive}(\hat{Y})$.

Therefore, in this subsection, only the gradient-based visualization results of images with high Dice scores using the *positive prediction* loss function were considered, which can be found in Figures 5.13 to 5.15.

From the visualization results, the first thing we noticed is that the gradients values of PET channel were usually ten times larger than the ones in the CT channel (for example Figure 5.13). However, this is understandable as the intensity values of the CT channel were between -100 and 100 **H**ounsfield **U**nits (HU) [49] while the PET channel contains pixels with values between 0 and 20 of **S**tandardized **U**ptake **V**alue (SUV) [50]. Another difference between the CT and the PET channel is that the areas that were important for the outputs of the Unet in the CT channel were noisier, more “scattered” and not as clear as the ones in the PET channel, even in the guided backpropagation results, which should, in theory, provide sharp and clear results (Figure 5.15).

In both the CT channel and the PET channel, pixels at the edges of the tumors were more important for the predictions than pixels at the inner parts of the tumors as shown in Figures 5.13 to 5.15. In both saliency maps and guided backpropagation results shown in Figures 5.13 and 5.15, pixels with high correlation to the prediction were the ones at the edge of the lymph nodes, while pixels in the inner parts of the lymph nodes were not as important. Some pixels around the tumor and in the center of the tumor had small negative impacts. However, in guided backpropagation, the negative gradients are the negative gradients when propagating from the first convolutional layer to the input layers. Therefore, negative gradients in guided backpropagation results did not have any significant impacts on the predictions of the model.

The deconvnet results, on the other hand, had opposite results relative to the saliency maps and guided backpropagation results in Figure 5.14. Pixels at the edge of the lymph nodes were still important, but had negative gradients. Nevertheless, the deconvnet did not calculate the true gradients, but the imputed version of the gradients. Negative gradients in deconvnet did not mean negatively correlation with the loss score.

Since the saliency maps showed both positive and negative correlation of pixel importance and guided backpropagation gave sharper results, our interpretation hereafter will focus on these two methods. Deconvnet will not be discussed further. Nevertheless, results from all three visualization methods associated with the two loss functions (*positive prediction* and *true positive*) defined in Section 4.2.3 on page 45 can be found in Appendix F.

Images with Dice scores of 0.5

The pixel contribution to the delineation predictions, which were the gradient-based visualization results using the *positive prediction* loss function, for slices with a Dice score of about 0.5 were similar to the results of images with high Dice scores. The similarity was noticeable as the most important pixels were at the edge of the predicted tumor (Figures 5.16 and 5.17). In the saliency maps (Figure 5.16), pixels that contributed to the predictions scattered over both the predicted tumors and the ground truth area. One odd thing is that some pixels with high contribution were located at the background of the images, which contained no useful information. On the other hand, in the guided backpropagation visualization results, pixels having high impact on the outputs formed a clear ring-shape around the predicted tumors in the PET channel (Figure 5.17).

In the saliency maps using the *true positive* loss function (Figure 5.18), pixels in the false positive areas had significant negative correlation to the loss score, which means decreasing the intensity values at these areas may improve the accuracy of the predictions. In contrast, pixels in false negative areas had positive gradients, which means increasing the intensity of these pixels may result in more accurate predictions. The guided backpropagation results using the *true positive* function (Figure 5.19) were similar to the saliency maps, but sharper and less noisy than the saliency maps.

Images with low Dice scores

The results of gradient-based visualization for images with low Dice scores were in two extreme categories. For visualization results using the *positive prediction* loss function, the parts with high influence on the prediction were the same with the tumor predictions (Figures 5.20 and 5.21). On the other hand, in the visualization results using the *true positive* loss function, the parts which should be the ground truth had small contribution, which resulted in the inaccuracy in the predictions of the model (Figures 5.22 and 5.23).

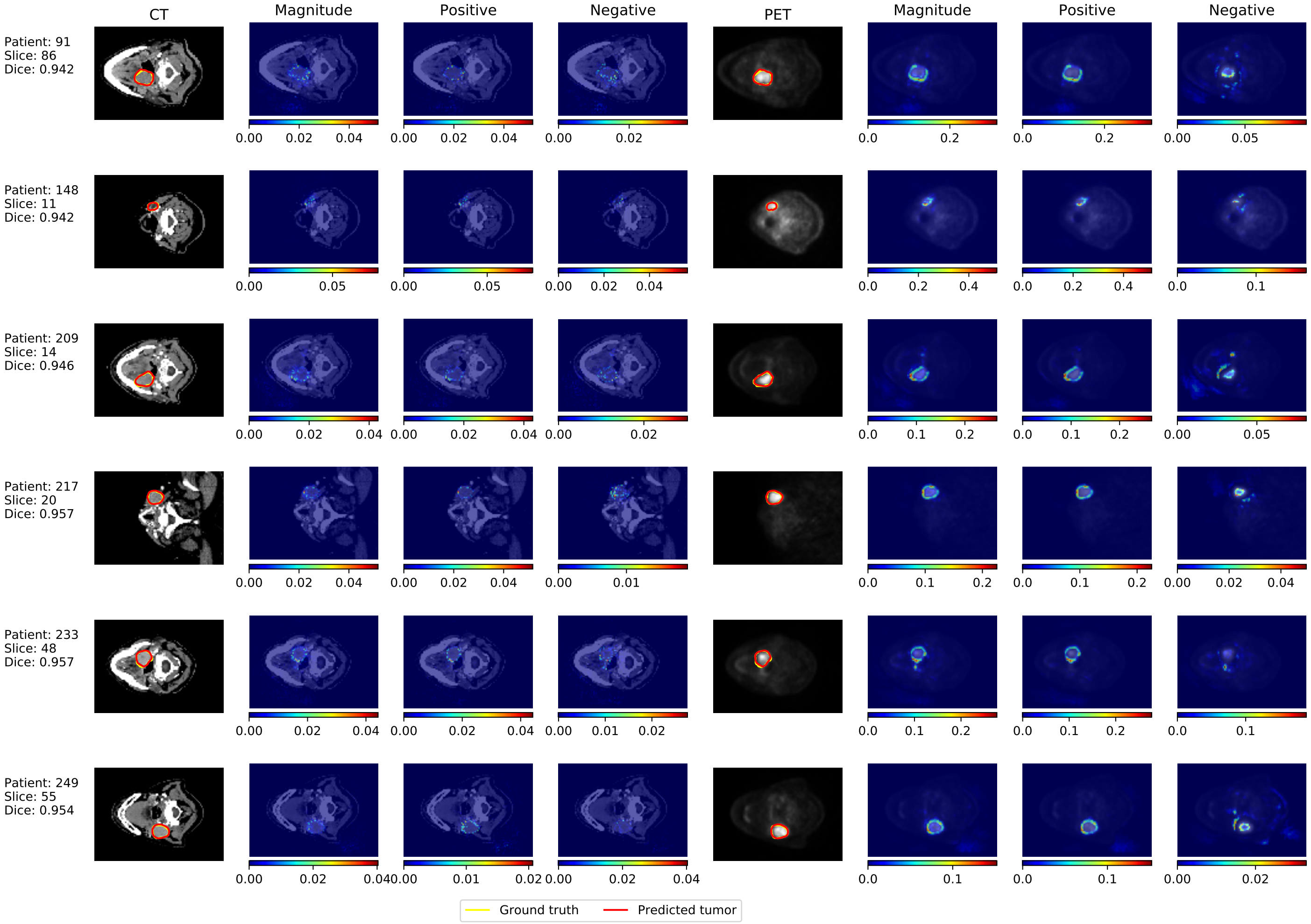


Figure 5.13: Saliency maps using *positive prediction* loss function of images with high Dice score. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan.

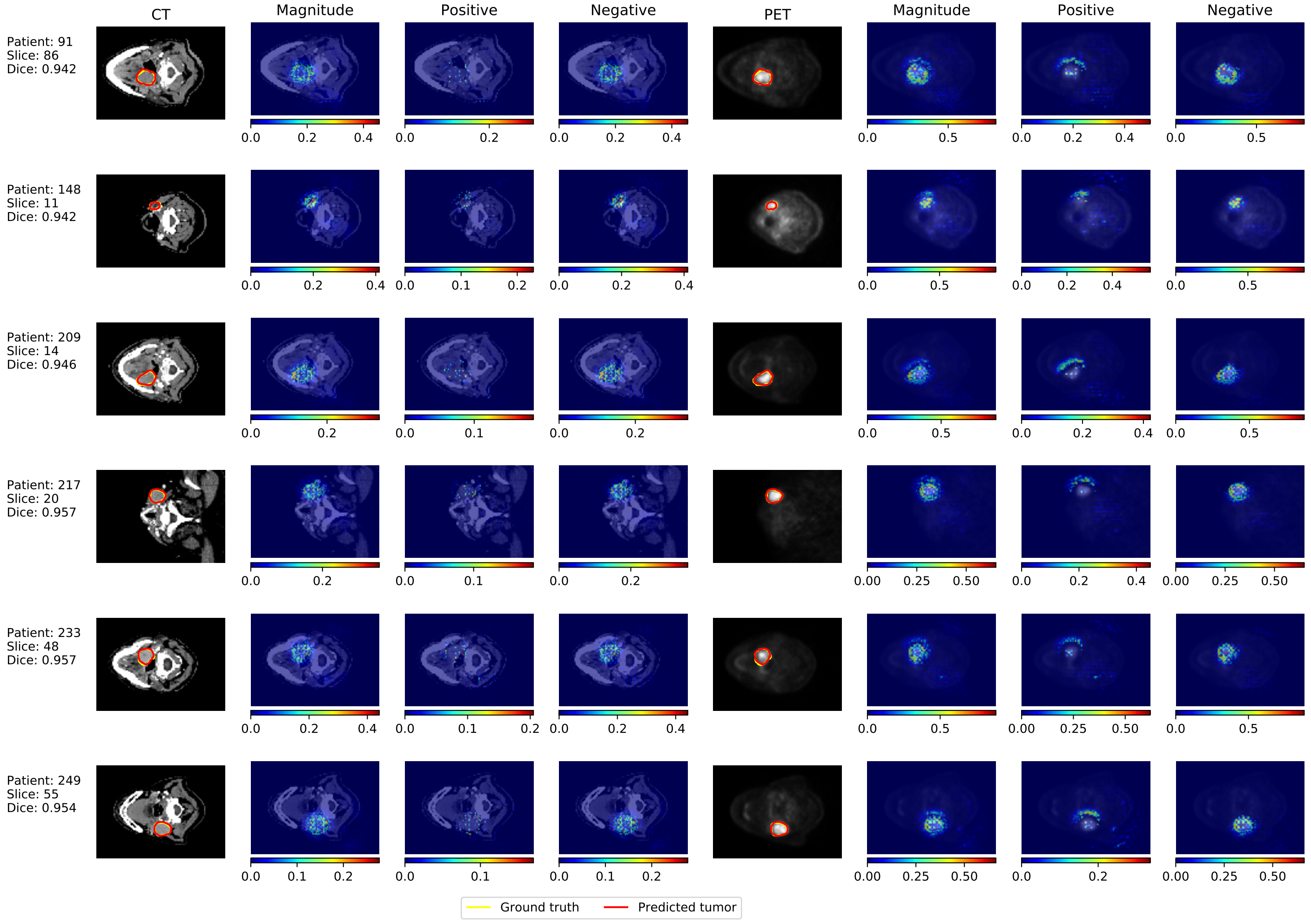


Figure 5.14: Deconvnet-based visualization results using *positive prediction* loss function of images with high Dice score. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan.

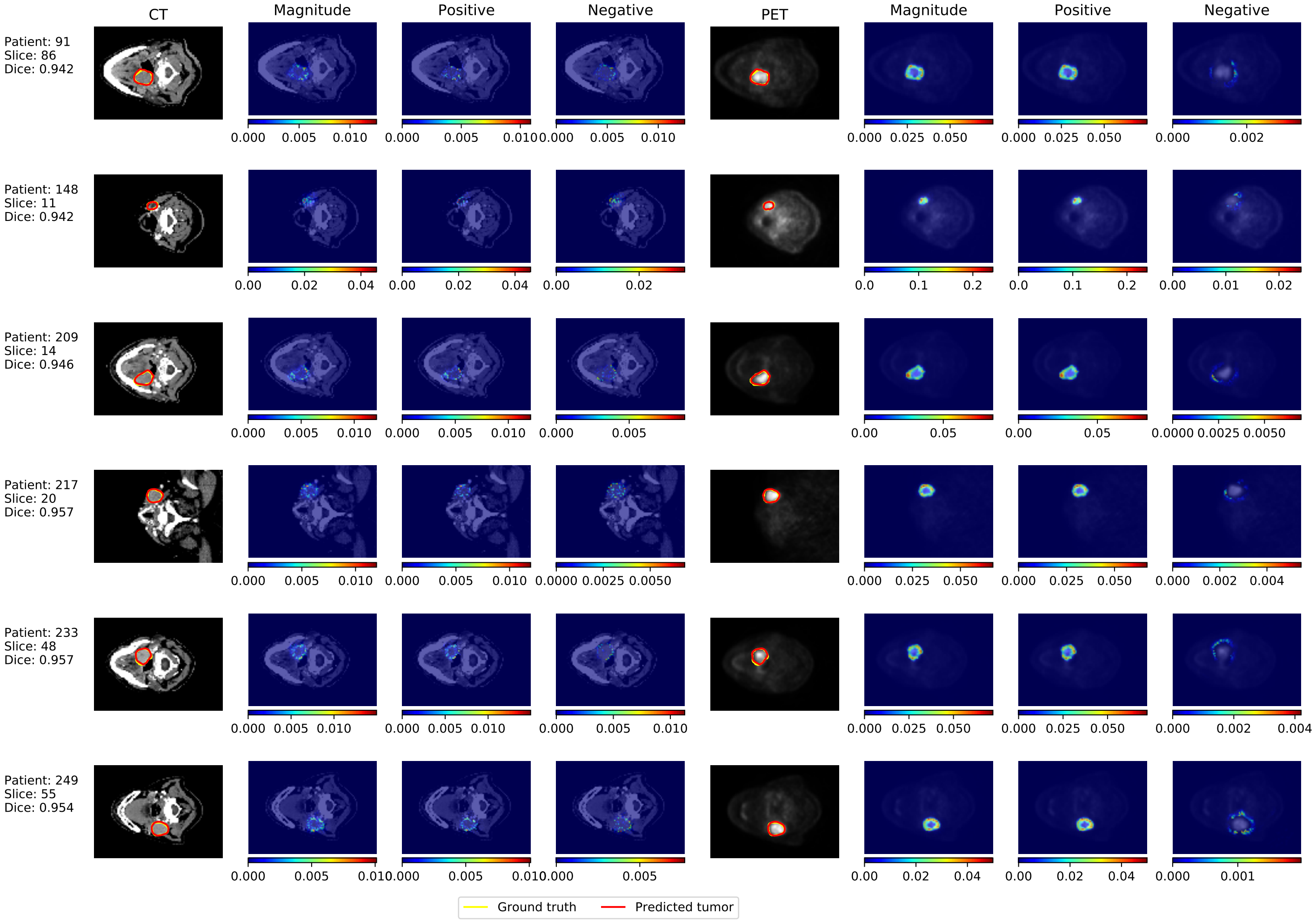


Figure 5.15: Guided backpropagation-based visualization results using *positive prediction* loss function of images with high Dice score. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan.

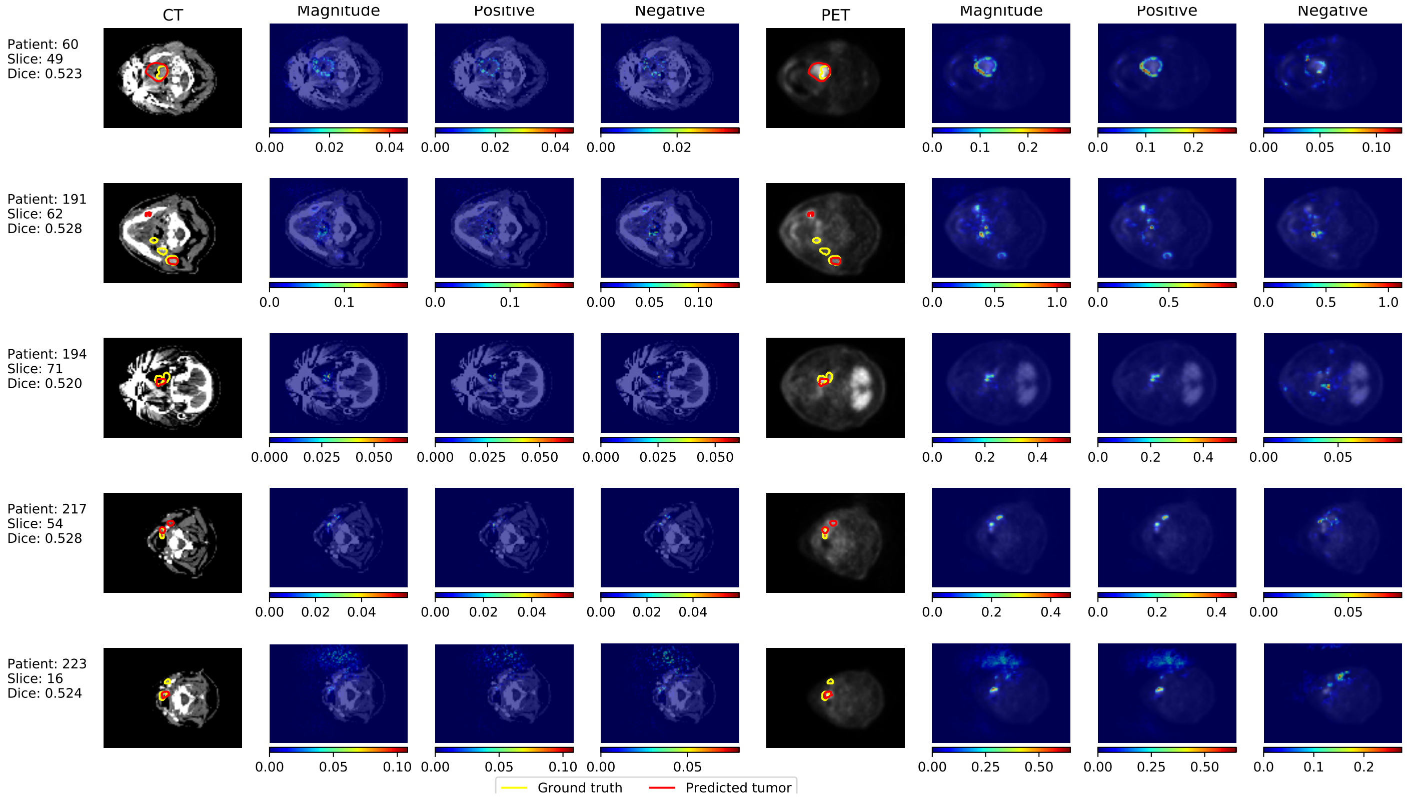


Figure 5.16: Saliency maps using *positive prediction* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

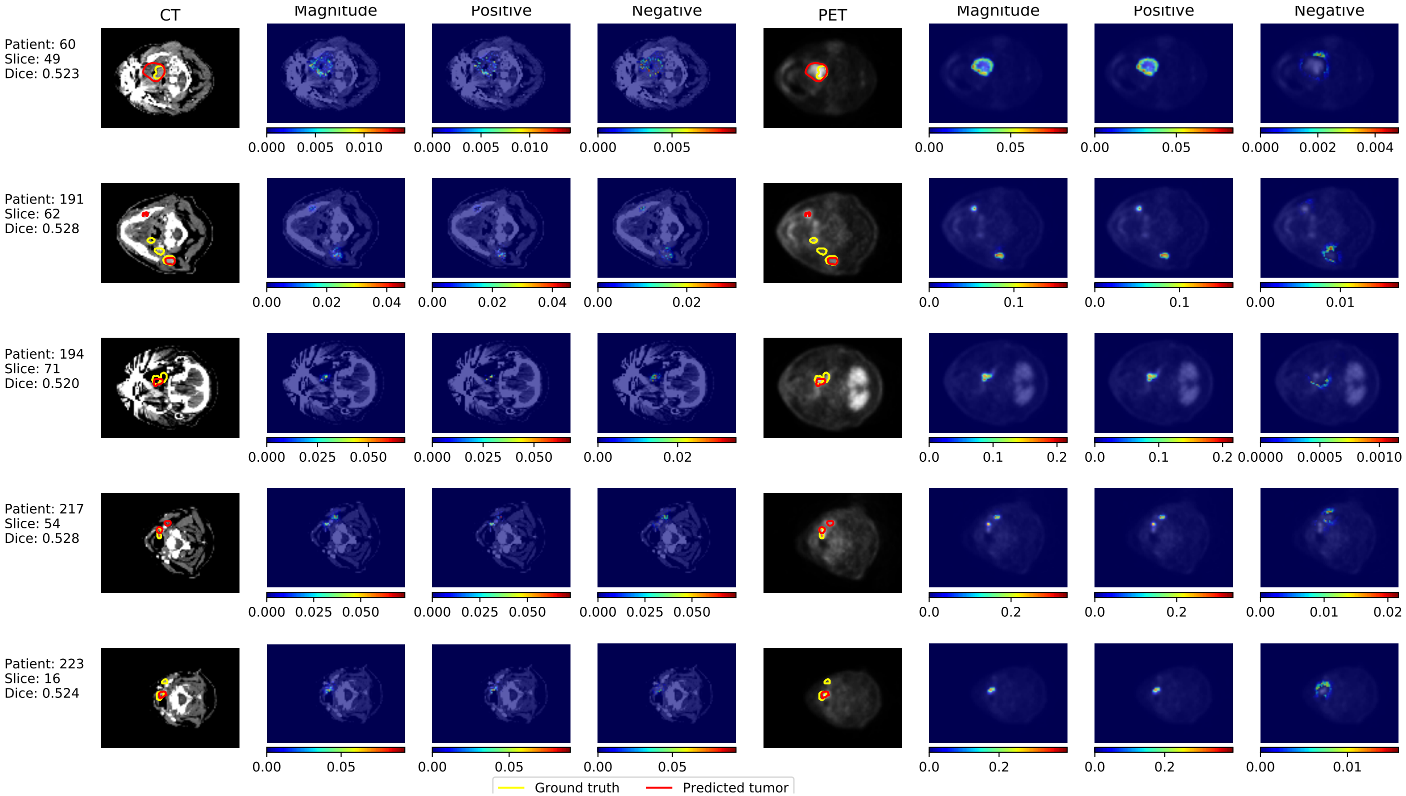


Figure 5.17: Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

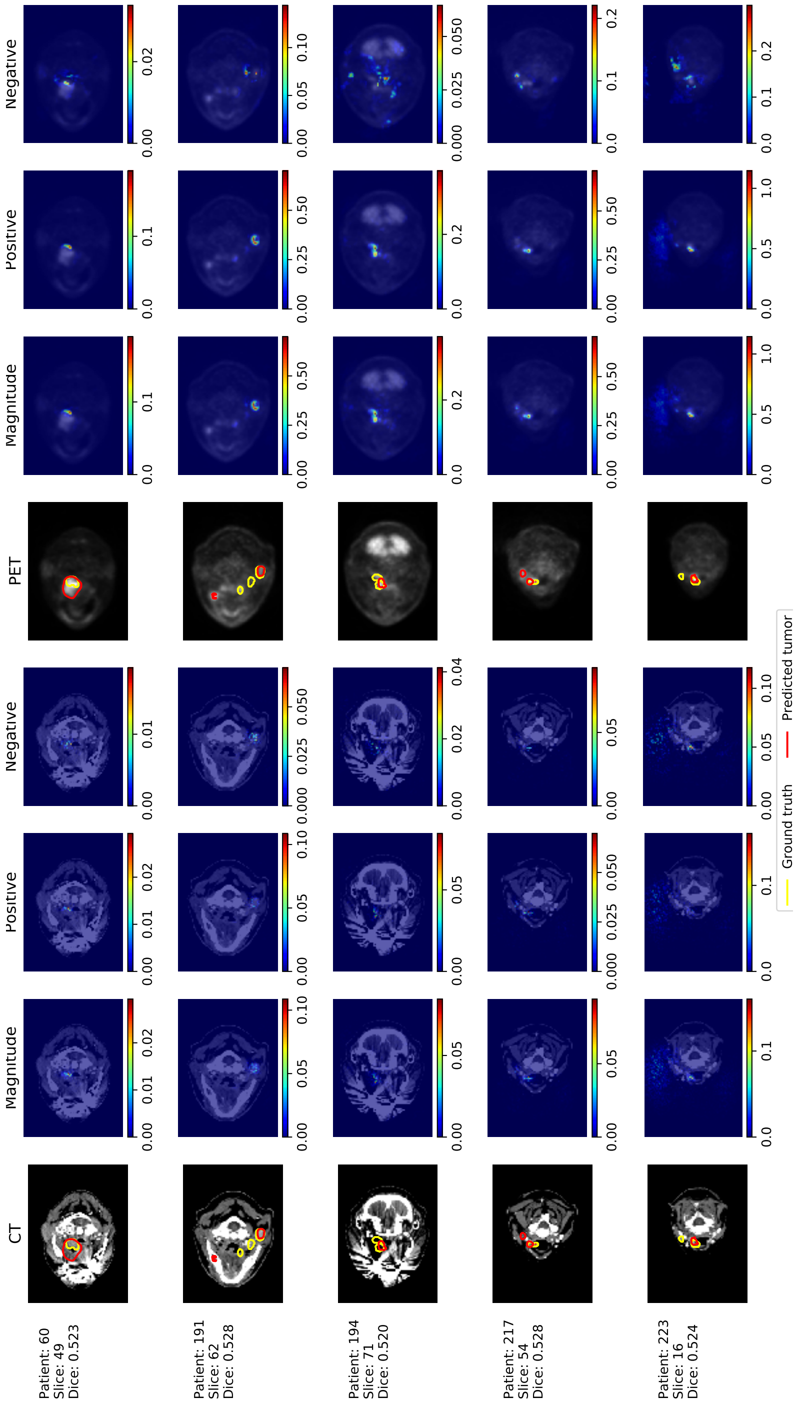


Figure 5.18: Saliency maps using *true positive* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

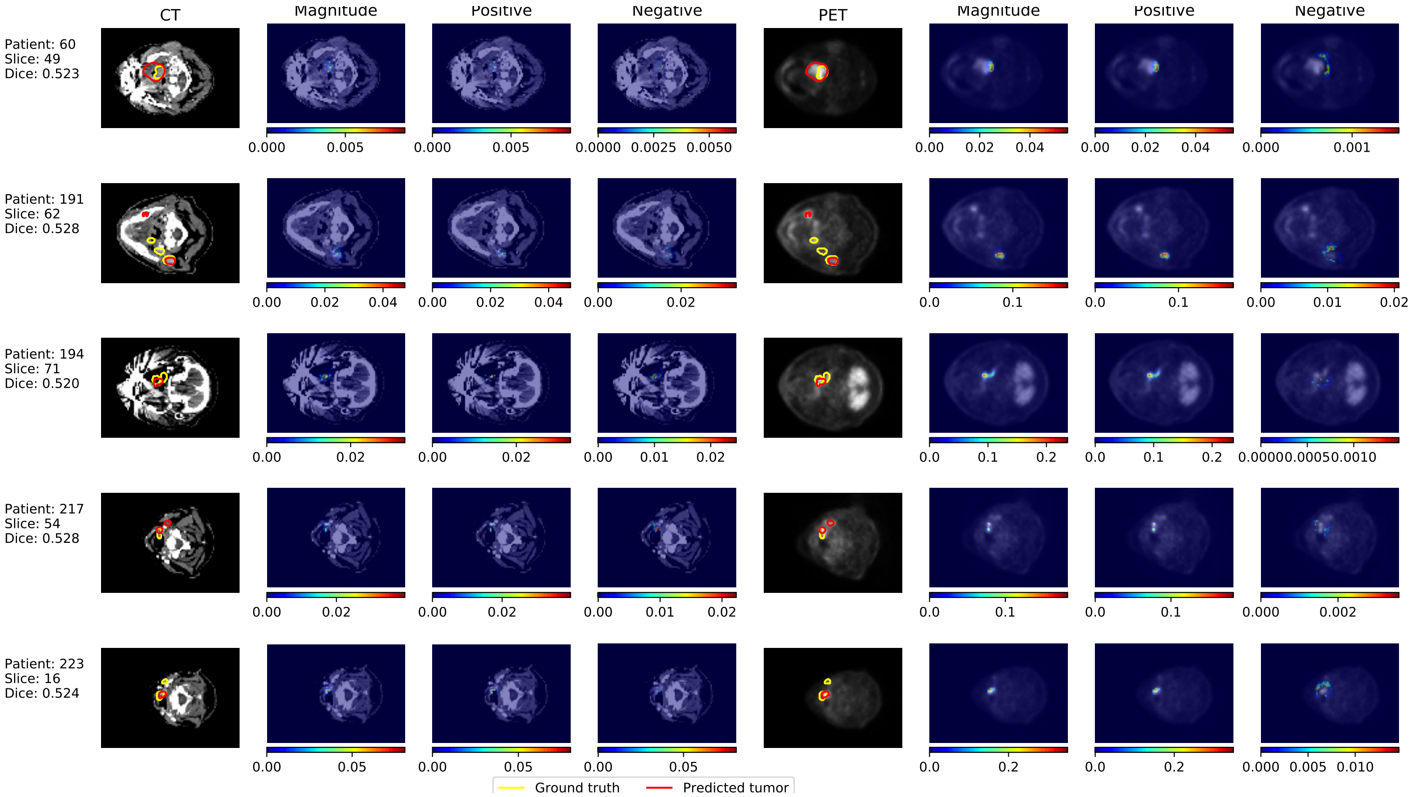


Figure 5.19: Guided backpropagation-based visualization results using *true positive* loss function parts of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

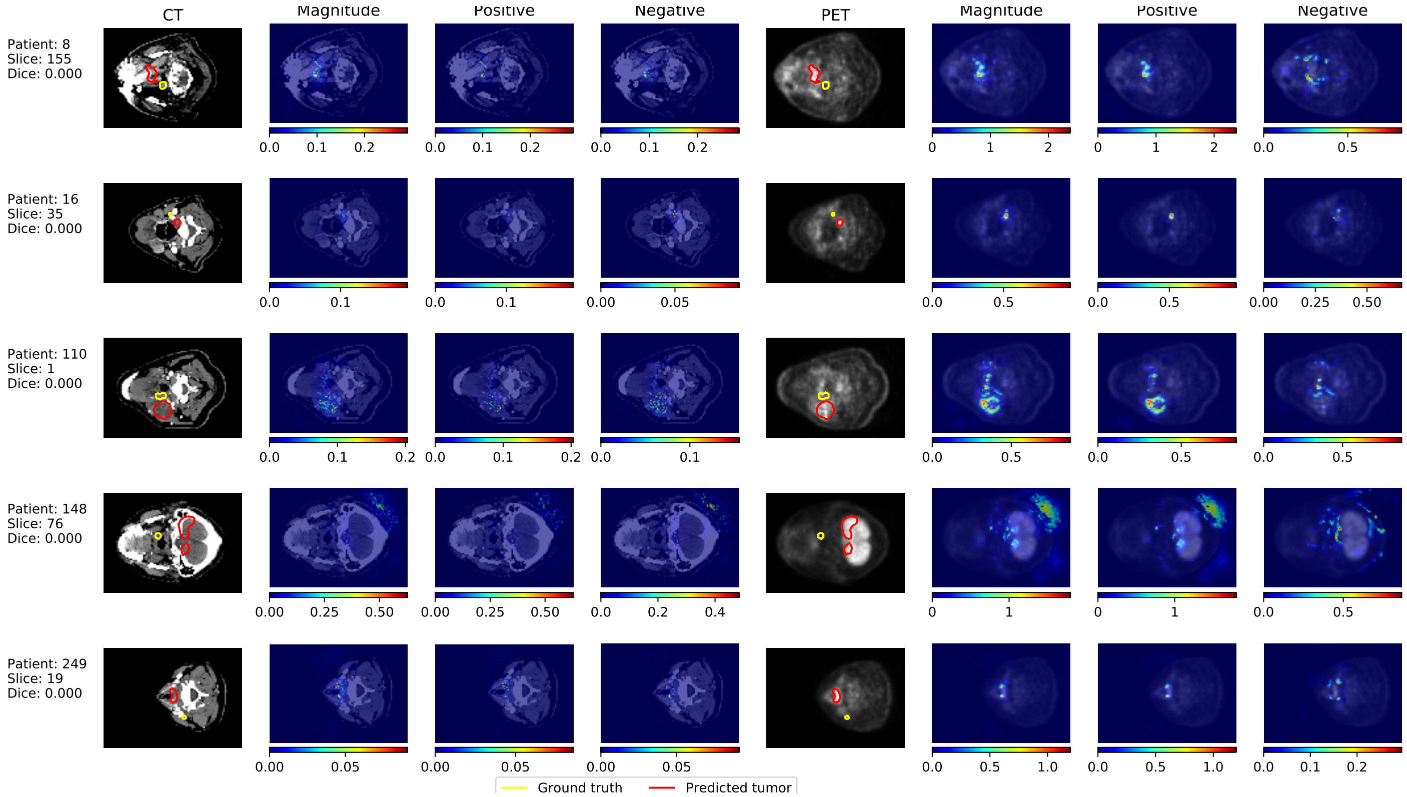


Figure 5.20: Saliency maps using *positive prediction* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

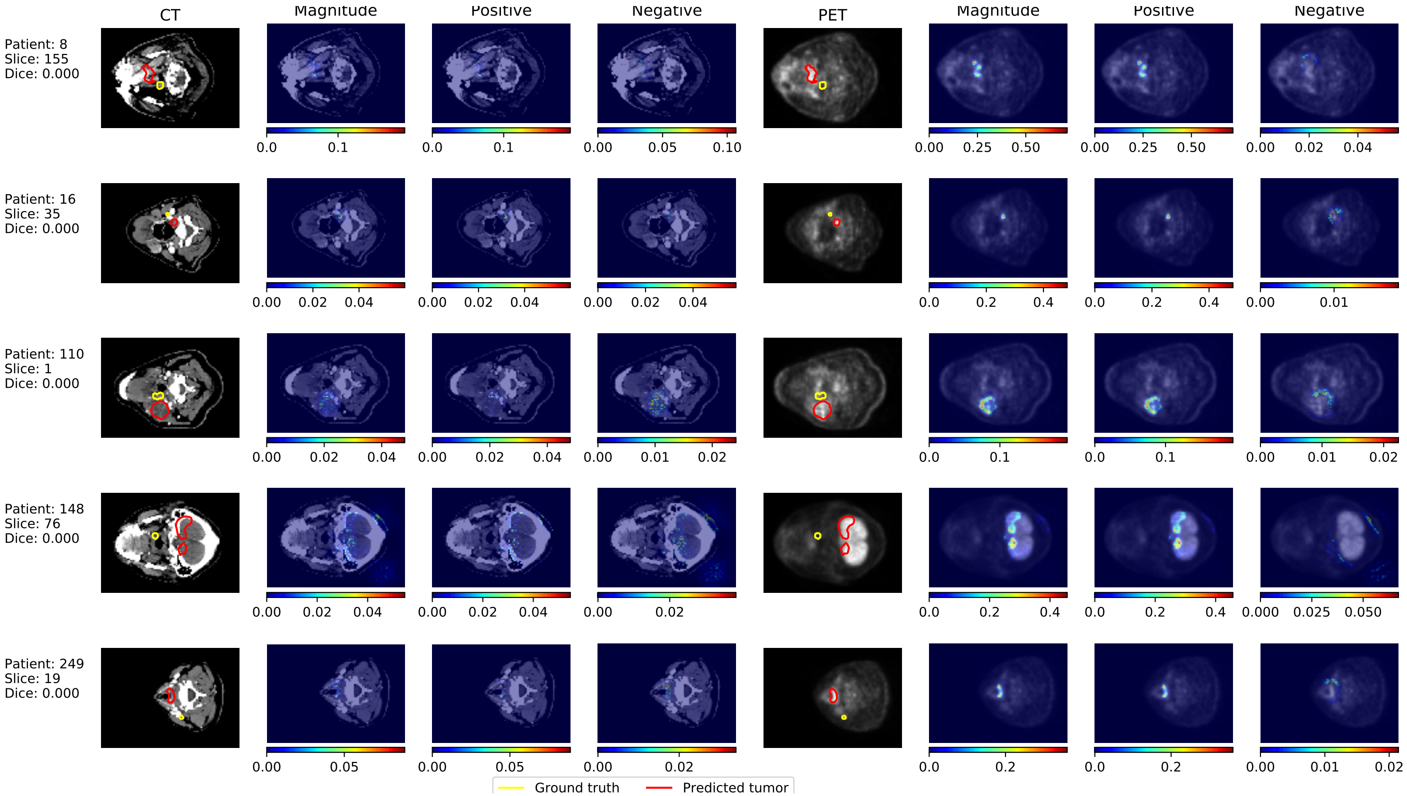


Figure 5.21: Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

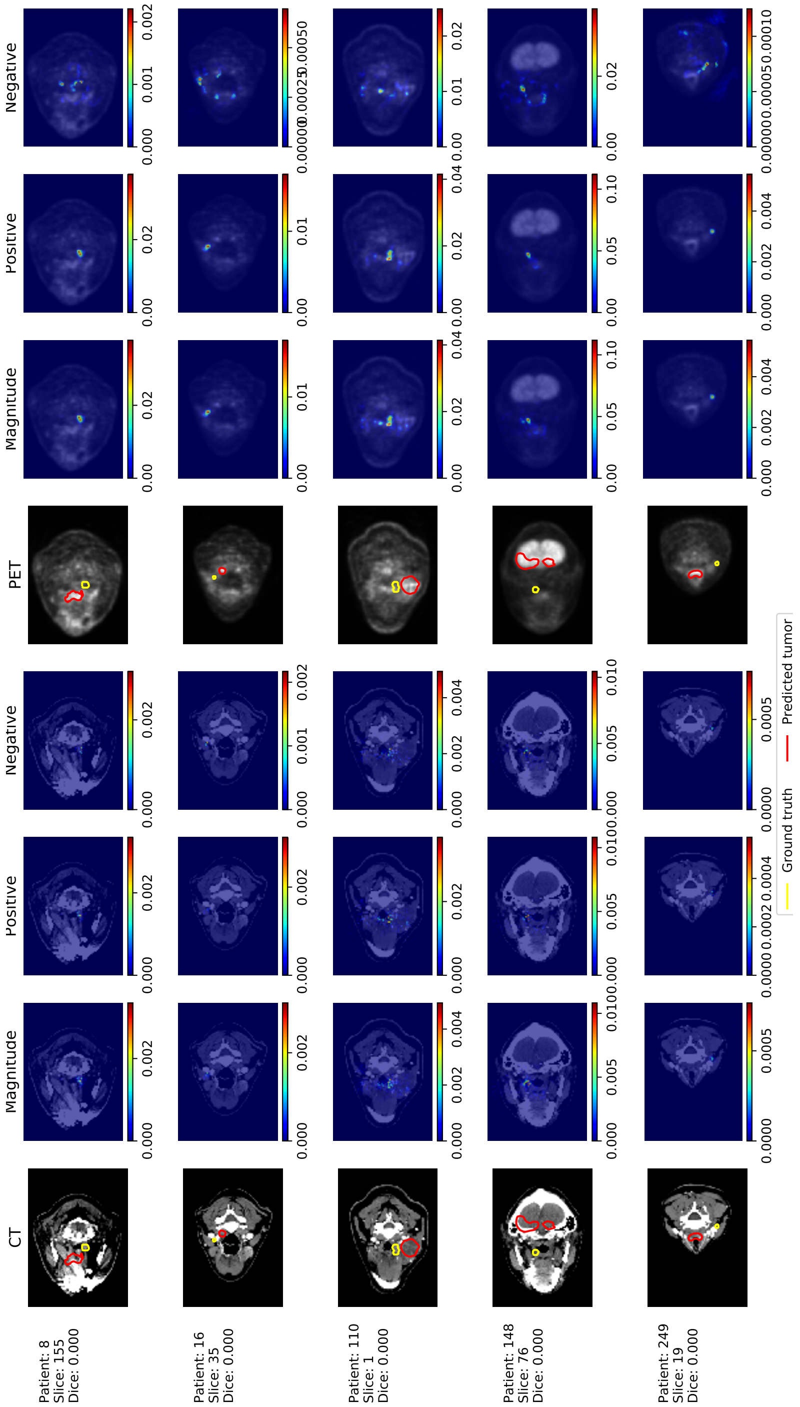


Figure 5.22: Saliency maps using *true positive* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

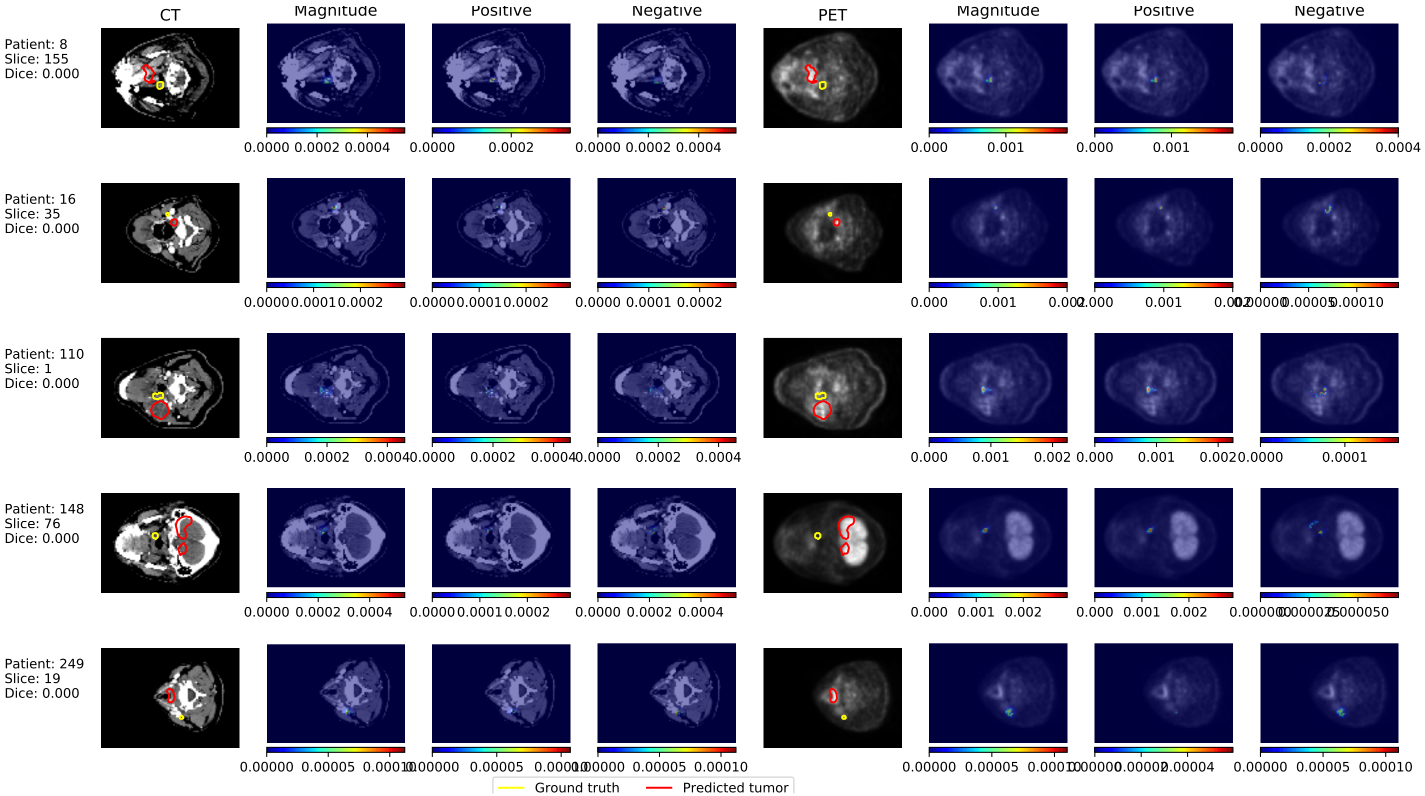


Figure 5.23: Guided backpropagation-based visualization results using *true positive* loss function parts of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

Chapter 6

Discussion

The developed *deoxys* framework contains the modules to perform deep learning experiments for solving different classification and segmentation problems. This framework covers all steps needed when searching for the best deep learning model. In this Master's thesis, the framework was updated to support different model visualization methods and integrate a database for managing the experiments.

The *deoxys* framework has a high level of reliability, as the process of running a deep learning experiment, as well as the newly integration of a DBMS, was tested manually and automatically. In addition, this framework was assessed by reproducing the results from previous studies about delineating tumor using deep learning model and visualizing the model.

This chapter discusses the visualization results of the Unet model (Figure 4.1) used for head and neck cancer tumor segmentation. From the interpretation results, the benefits of the model visualization for radiologists and data scientists can be demonstrated. Moreover, limitations of the *deoxys* framework will be discussed in this chapter, as well as the potential improvement for further development of this framework.

6.1 The Unet model

This section proposes possible solutions to improve the performance of the trained Unet model (Figure 4.1) based on the model visualization results.

6.1.1 Unet Model performance

The segmentation results of the Unet model (Figure 4.1) had an average Dice score of 0.638 and a median Dice score of 0.779, which may not good enough for use in the clinic. However, when comparing with other studies, we can see that this result did not significantly underperform as shown in Table 6.1. Because of the limited number of studies involving in deep learning for head and neck cancer tumor segmentation, similar studies using deep learning for tumor segmentation in other sites were also considered in this section.

In Table 6.1, we can see that the results of the Unet model trained using the *deoxys* framework were similar with other models trained on PET/CT images for head and neck cancer segmentation. Other models, except for the DeepMedic model proposed by Kamnitsas *et al.* [51], outperformed the Unet model used by the *deoxys* framework. However, these models were either segmentation of one specific type of head and cancer tumor, such as Nasopharynx cancer [52][53] or Oropharyngeal cancer [54], or segmentation of brain tumors. Therefore, the Unet model used in this Master's thesis had an acceptable performance relative to other studies.

Name	Description	Avg. DS/slice	Median DS/slice	Avg. DS/ patient	Median DS/ patient
deoxys	2D Unet on PET/CT scan	0.638	0.779	0.733	0.775
Huang *et al.* [10]	2D Modified UNet on PET/CT scan	N/A	N/A	0.720	N/A
Guo *et al.* [55]	3D Dense-Net [55] on PET/CT scan	N/A	N/A	0.71	0.73
Guo *et al.* [55]	3D Unet on PET/CT scan	N/A	N/A	0.69	0.71
Lin *et al.* [52]	3D VoxResNet [56] on MRI scan (Nasopharynx cancer)	N/A	N/A	N/A	0.79
Cardenas *et al.* [54]	3D Unet on CT scan (Oropharyngeal cancer)	N/A	N/A	0.815	0.817
Men *et al.* [53]	Deep deconvolutional neural network [53] on CT scan (Nasopharynx cancer)	N/A	N/A	0.809	N/A
Kamnitsas *et al.* [51]	DeepMedic [51] on MRI scan	N/A	N/A	0.630	N/A
Natekar *et al.* [57]	2D DenseUnet [57] on MRI scan	N/A	N/A	0.830	N/A
Natekar *et al.* [57]	2D ResUnet [57] on MRI scan	N/A	N/A	0.788	N/A
Natekar *et al.* [57]	2D SimUnet [57] on MRI scan	N/A	N/A	0.743	N/A

Table 6.1: Comparison of the results of tumor segmentation between the *deoxys* framework and other studies using the Dice score (DS). The first group contains the results of the head and neck cancer tumor segmentation. The second group contains the results of specific types of head and neck cancer tumor segmentation. The final group contains the results of brain tumor segmentation.

6.1.2 Model visualization results

Activation maps

It is difficult to evaluate the activation maps results of the Unet model trained on the head and neck cancer dataset used in this Master's thesis as there are no deep learning studies using this visualization method on the head and neck cancer data. However, interpreting the Unet model using activation maps is still possible with the existing results.

When visualizing the activation maps in Section 5.2.1 on page 53, there were a high rate of similarity between the features extracted by different filters in the same layer. This may be because of (1) the complexity of the Unet model, which caused redundancy in some of the filters, or (2) the insufficiency of model training to utilize all of the filters in the model.

For the first case, decreasing the complexity of the model by having fewer filters in each layer may provide a simpler model, which can reach the same performance as in Table 6.1. In addition, this modified simple model would occupy less space in the memory of the computing environment. Moreover, the training process using a simpler model would also be faster due to the decreasing number of data computations. Another way to improve the model is to add Dropout layers [58], which zero out some of the nodes in CNN to prevent overfitting.

For the second case, continuing to train the model may increase the performance of the model. In addition, training the model with more diverse images created by data augmentation [59][60] could enable the model to utilize more of the filters.

Data augmentation is a deep learning technique to generate different forms of the input data to prevent overfitting [60]. Data augmentation involving in image data is transforming the input images into multiple versions of these images. This process includes scaling, rotating, flipping, adding noises and jitters, blurring, etc, to the images [61].

To assess the proposed solutions, we can look at the changes in activation maps to check for the increment of features extracted in the newly trained model.

Activation maximization

The results of the activation maximization method provided limited information, as most of the generated images were noisy and abstract (Figures 5.11 and 5.12). This may be because (1) most filters in the activation maps searched for the lymph nodes in the PET channel and other filters extracted parts with similar intensity values in the CT channel. Thus, these filters may mostly be frequency passing filters, which search for some specific ranges of intensity values and may not looking for any kinds of patterns. This can be resolved by continuing training the original model.

Another reason may be because (2) the input images lacked data augmentation, such as rotations, blurriness. Since all images were trained in the same direction, the layers were not forced to search for specific shapes or patterns in the images. Thus, after training the Unet model with data augmentation, there could potentially be more meaningful patterns that the filters in the Unet model search for.

One final reason, which is unrelated to the performance of the Unet model, may be because (3) the implementation of the activation maximization method needs some advanced techniques so that this method can yield more meaningful results. Natekar *et al.* [57] successfully generated less noisy images using activation maximization in a Unet-based model by applying regularization, jitters and total variance to the process of visualization using the activation maximization methods. Therefore, applying these techniques in the *deoxys* framework may improve the results of the activation maximization method.

Gradient-based visualization

In the results of the Unet model's gradient-based visualization (saliency maps, deconvnet, guided backpropagation), the areas with high contribution to the model predictions in the PET channel were clearer than in the CT channel (Figures 5.13 to 5.23). Note that this does not referred to the difference in intensity values between the CT channel and the PET channel, but rather how much the areas with high impact stood out from the background of the gradient-based visualization results. Thus, we can say that the PET channel had higher impact on the segmentation results than the CT channel (1).

In the gradient-based visualization results of the images with high Dice score, the pixels at the edge of the lymph nodes are more important for the predictions (Figures 5.13 and 5.15). This shows that the model searched for significant difference between neighboring pixel values when making a prediction (2).

Points (1) and (2) above lead to a hypothesis that the Unet model (Figure 4.1) learned that when "bright dots" existed in the PET channel, there is a high chance that these bright areas in the PET channel are cancerous. Note that this hypothesis does not deny the contribution of the CT channel, as the CT channel still had some impacts on the model outputs, but not as much as the PET channel.

This hypothesis is supported by the fact that most of the images with high Dice score had PET channels with clear and bright lymph nodes while the images with intermediate and low Dice scores had PET channels with low intensity values and unclear lymph nodes (Figures 5.13 to 5.23). In addition, from the gradient-based visualization results of images with intermediate and low Dice score, pixels that contributed to the false positive areas also had high intensity values in the PET channel (Figures 5.16, 5.17, 5.20 and 5.21), which also strengthens the stated hypothesis. However, the predictions of the model did not completely depended on the lymph nodes in the PET channel, as the PET scan at slice 11 of patient 148 did not contain any areas with high intensity values but the prediction got a high Dice score (Figure 5.13).

As the gradient-based visualization methods were proposed for classification problems [18][19][20], there is lack of studies about deep learning in head and neck cancer segmentation using these methods to support the above hypothesis. However, a study in colorectal cancer used a modified guided backpropagation method to visualize the important pixels when delineating colorectal polyps [62]. This study shows that the pixels at the edge of the polyps had high impact on the predictions of the model, which is the same as the importance of the edge of the lymph nodes for the prediction of the Unet model (Figure 4.1) in this thesis.

For the images with intermediate and low Dice scores, the gradient-based visualization results, especially the saliency maps, shows that the model was confused when delineating cancer tumors from these images. This is evident in the high contribution of the pixels scattered in the true positive and false positive areas, and in the unrelated parts of the images (the background) (Figures 5.16, 5.18, 5.20 and 5.22). Thus, the noisiness and uncertainty in the Unet model's gradient-based visualization results indicated that this Unet model still has potential for improve, as the gradient-based visualization results were not good enough. This is because the parts in the saliency maps indicating the important pixels should be clearer comparing to the background when the model's performance is optimal, as shown in [63].

As the model predictions heavily depended on the PET channel, one approach to improve the performance of the Unet model is to increase the contribution of the CT channel by blurring the PET channel in the data augmentation process. Another data augmentation technique that is applicable in this approach is to increase the contrast of the CT channel. We can look at the saliency maps or the guided backpropagation results of the newly trained model to check for the improvement of the performance.

6.2 Benefits of model visualization

This section explains how model visualization of deep learning increases the transparency and interpretability of the CNN, which benefits both radiologists and data scientists.

6.2.1 For radiologists

Thanks to the results of activation maps of the Unet model (Figure 4.1) in Section 5.2 on page 53, the segmentation results from deep learning are no longer black-boxes to radiologists. With the activation maps, the deep learning model can provide insights for its decisions. These insights contain human-understandable information illustrated in the filters of the activation maps.

As the same filter at a specific layer extracted the same features and patterns from different input images (see Figure 5.10), the activation maps can also be used as “feature extractors”. When a deep learning model has been sufficiently well trained that each filter in the convolutional layers is specialized to one or more parts of the image data, radiologists can search for abnormalities in the images by directly looking at the results of one or more filters. These types of feature extractors are especially useful for features with low intensity values or that are blended in perfectly in some area of the images with only small intensity values.

With the explanation from gradient-based visualization results, radiologists can look at the pixels in the original images that influence the predictions of the deep learning model. These visualization results also show the reliability of the model predictions as robust models give gradient-based visualization results with high interpretability [64][65]. This property can be seen in the Unet model from the unreasonable gradient-based visualization results when incorrect delineation results occurred as shown in Figures 5.16 to 5.23 and as discussed in Section 6.1.2. Based on the suggestions and explanations of the deep learning model, radiologists can decide whether to approve, discard, or modify the results from the deep learning model.

6.2.2 For data scientists

Many solutions were proposed to improve the Unet model based on the model visualization results in Section 6.1.2. Therefore, in addition to the performance analysis of the CNN, data scientists can use the model visualization results to explain the behavior of the proposed model, as well as proposing possible approaches to improve the performance of that model.

To be specific, with the use of activation maps and activation maximization, data scientists can understand how the deep learning model processes the data, together with the patterns the convolutional layers extract [66]. From the interpretation of this information, data scientists can find weakness of the model, and can thus easily resolve the problems. In addition, with the help of model visualization, data scientists can monitor the improvement of the model when applying their solutions to improve the model.

The results from gradient-based visualization, which show the important part of the input images, also help data scientists to understand which parts of the images have high impacts on the prediction of the deep learning model. In addition, the results of these methods provide the robustness of the proposed model, as Zheng *et al.* [63] used the saliency maps to check for the effectiveness of their proposed model. Moreover, based on the gradient-based visualization results, data scientists can find approaches to preprocess the input data so that the deep learning model has a higher probability of making the right predictions.

6.3 Application to other cancers

The *deoxys* framework was designed to solve any deep learning problems. Thus, as the Unet model (Figure 4.1) trained by the *deoxys* framework reached an acceptable performance as shown in Table 6.1, the *deoxys* framework should be applicable for tumor segmentation in other types of cancers.

The benefits of model visualization discussed in Section 6.2 are not limited to the Unet model trained on the head and neck cancer dataset. In other words, these model visualization methods are applicable for any other CNNs. Apart from Natekar *et al.* [57] and Wickstrøm *et al.* [62] who used visualization methods to explain, interpret and evaluate their models in tumor segmentation problems, there is a lack of studies using model visualization methods in segmentation problems. Nevertheless, CNNs in classification problems were able to be assessed using model visualization results [63][64][65][66]. Therefore, radiologists and data scientists working on other types of cancers can also benefit from applying deep learning model visualization on their works.

6.4 Difficulty in the implementation process

Implementation of gradient-based visualization methods, especially the deconvnet method and the guided backpropagation method faced some difficulty. In the deconvnet and the guided backpropagation method, the imputed versions of the gradients of the ReLu function were calculated (see the equations in Section 2.3.3). However, it was impossible to change the algorithm for calculating the gradients directly in Keras [23]. This is because Keras only works as the top-level interface of neural network components while this kind of algorithm belongs to the Keras's deep learning backends, which contain different algorithms for efficient data computation in the computing environment. Therefore, the Keras's backend, Tensorflow [39], was used in this implementation. The Tensorflow backend enabled registering new algorithms for calculating the gradients of the ReLu function.

Because of the shifting of the Tensorflow backend from 1.x version to 2.x version since last year (2019) [67], the implementation for registering the new gradient-calculating algorithms was forced to be compatible with both versions of Tensorflow. This required further research as Tensorflow 2.x was just released a few months ago with huge updates in comparison with Tensorflow 1.x.

Another difficulty was that saliency maps, deconvnet and guided backpropagation were original proposed for classification problems. Although these methods can be generalized into segmentation problems, or other deep learning problems, by using the right *loss function* in Figure 2.10, there were not any good examples of these methods in deep learning problems other than classifications. This slowed down the implementation process as researches and tests were performed in this thesis to ensure the reliability of these visualization methods in the *deoxys* framework.

6.5 Potential for improving the deoxys framework

6.5.1 Improvement of existing visualization methods

Medical images data do not have clear patterns for a neural network to learn [57]. As a result, it is difficult to generate the patterns the filters search for by using the activation maximization method with the basic implementation. With the addition of jitters [68], the total variation regularizer [69], and learned priors [17], the activation maximization method has a better opportunity to generate meaningful images.

Jitters [68] happen before the adjustment of the initial image with random noise in Figure 2.8 in every iteration. When applying jitters, that image is cropped by randomly cropping by a specific size, then is upscaled back to its original size. This process results in an image with random noise that correlates with the neighboring pixels.

The total variation regularizer also happens before the image with random noise in Figure 2.8 is updated in every iteration. Unlike jitters, the total variation regularizer [69] smooths the image while maintaining the edges of the objects in that image. This process results in a blurrier image with sharp edges.

A more complicated technique that can be used in the activation maximization method is to apply constraints, which is called learned priors [17], when normalizing the gradients to adjust the image in Figure 2.8 in every iteration. One example of these constraints is that the resulting image should have similar properties with a natural image, i.e., the neighboring pixels are correlated.

When implementing the gradient-based visualization methods, modification of gradient calculation was applied to the ReLu function. However, the ReLu function is not the only option for the activation function for hidden convolutional layers. The LeakyReLu function [70] is a modified version of the ReLu function where the negative linear combination values of the nodes in the neural networks were downscaled instead of silenced (see the below equation).

$$\phi_{LeakyReLu}(x) = \begin{cases} \alpha \cdot x & \text{if } x < 0, \\ x & \text{otherwise} \end{cases} \quad \text{for } 0 < \alpha < 1$$

This type of activation function was proposed to solve the problem of the "dead" ReLu activation units, when there are no weight adjustments because of the high occurrence of negative nodes before the ReLu function [70]. Models using the LeakyReLu activation function sometimes yield better results than the ReLu function [71].

The current implementation of deconvnet and guided backpropagation in the *deoxys* framework involved registering a "new" ReLu function with a different approach to calculate the gradients. This new function then replaced the original ReLu function in the model so that the gradients were calculated using the equations described in Section 2.3.3, which explains the gradient-based visualization methods.

When the model contains the LeakyReLu activation function, it should replaced this function with the "new" ReLu function above when applying gradient-based visualization methods[1]. Unfortunately, the *deoxys* framework did not support gradient-based model visualization for models using other activation functions than the ReLu function[2]. Therefore, making deconvnet and guided backpropagation to support model using activation functions derived from the ReLu function was also necessary in the *deoxys* framework.

6.5.2 Addition of visualization methods

Saliency maps, deconvnet and guided backpropagation are not the only methods for determining the pixel importance of the input images. Other methods such as Class Activation Maps (CAM) [21] and Gradient-weighted Class Activation Mapping (Grad-CAM) [22] were proposed with similar goal.

Each filter of the activation maps of the last convolutional layer L_{conv} before the output layer contribute differently to the model prediction L_{out}. These contributions are in the form of the weight vector of the output layer W_{out}. Therefore,

---

[1] Springenberg *et al.* [20], who first proposed the guided backpropagation method, trained a model with the LeakyRelu activation function in [20], but then used the ReLu function when visualizing that model using gradient-based method.

[2] The activation function of the last layer was not include in this case, as the gradient-based visualization methods involve the activation functions of the hidden layers.

combining all filters in L_{conv} using the weights W_{out}, i.e calculating $L_{conv} \cdot W_{out}$, generates a heatmap representing the important areas of the input image to the model predictions. The process of creating this heatmap explains the CAM and Grad-CAM methods. Although the CAM was limited to CNNs with a global average pooling layer preceding the output layer, Grad-CAM was generalized for use on any type of CNNs.

Unlike the saliency maps, deconvnet and guided backpropagation visualization methods, which highlight the important pixels and generated images with sharp lines from important features, the CAM and Grad-CAM results provide zones that contributes to the outputs of the model. Because the convolution layer L_{conv} does not always have the same size as the original input image, scaling the resulting heatmap to overlay on the input image can create important regions that spread over the input image. Thus, the main difference between these two groups of visualization are that one group looks for clear and sharp patterns, while the other group searches for a region.

When the *deoxys* framework support the CAM and Grad-CAM methods, users will have a wider range of visualization options to choose between depending on the goal.

6.5.3 User interaction

With the use of a database for management and visualization methods for model explanation, an interactive user interface is essential so that non-tech people are able to use the *deoxys* framework easily. The user interface can be a web-based management application to manage the training experiments. In addition, a “click and play” user experience would be better for the users of the *deoxys* framework. This kind of user experiment allows the *deoxys* framework’s users to interact with the framework just by clicking in the computer to choose from various options.

Supporting an interactive user interface with “click and play” experience benefits the radiologists as they can use the framework directly without actually learning programming skills. In the case of data scientists, having the knowledge about deep learning model and the visualization methods is enough to use the framework. This would save them much time reading the documentations and instructions, especially data scientists with a lack of programming skills.

Ideally, when interacting with the *deoxys* user interface, it would useful for radiologists to have the following options when delineating cancer tumors.

1. Choose the medical image.
2. Check the segmentation results of the model.
3. Choose to view one or more filters from activation maps from a list of names and numbers. It would even better if they can have their "favorite" filters marked and ready to view for any images.
4. View the auto-generated performance analysis reports from the experiment.
5. Choose one or more visualization methods to view the explanation of the model for making that prediction. For radiologists, the default loss function is the "positive prediction" loss function.
6. Approve, discard or modify the prediction directly on the predicted image.

For data scientists with novice programming skills, the ideal user interface should allow them to handle the following processes by using the mouse and a few keyboard inputs.

1. "Drag and drop" from a list of components to configure a CNN model.
2. Run the experiments by just clicking "Run" and wait for the results. If possible, the data scientist can view the performance of the current training experiment while waiting for the results.
3. Choose to view one or more filters from activation maps and activation maximization from a list of names and numbers.
4. Choose to view one or more results from the gradient-based visualization methods from a list of predefined loss functions.

Chapter 7

Conclusion

The final goal of the deep learning framework, named *deoxys*, is to create a user-friendly software that can help radiologists with tumor delineation problems. To achieve this goal, the *deoxys* framework was designed and developed to perform deep learning experiments for automatic tumor cancer segmentation. This framework was generalized to work with different forms of image data and CNN architectures. In addition, this framework was designed to cover the steps of performing machine learning experiments to define, customize and find the best deep learning approach for different problems.

In this Master's thesis, model visualization for explaining the deep learning model was added into the *deoxys* framework. These included feature extraction methods such as activation maps and activation maximization, and gradient-based methods for finding the input part that has high influence on the model predictions and interpretation of model behavior such as saliency maps, deconvnet, and guided backpropagation. In addition, management of experiments using a database also integrated into the *deoxys* framework. These updates were assessed by the ability to reproduce results from previous studies for the case of model visualization supports and manual tests for the database integration.

The implemented visualization methods were applied to a convolutional neural network trained on head and neck cancer data of PET/CT images for cancer segmentation. By interpreting the model visualization results, we found interesting behavior of the pretrained model. From the activation maps, we found that many filters in the layers of the pretrained model extracted the tongue, bones (jaws and spine), muscles and glands from the CT scans and the lymph nodes from the PET scan. From gradient-based visualization results, we found that the model learned that there was a high probability of cancer tumors when bright lymph nodes in the PET scan existed. In addition, weaknesses of the pretrained model such as lack of data augmentation was found when interpreting the visualization results.

From the interpretation of visualization results of the pretrained model, we demonstrated how radiologists and data scientist could benefit from using model visualization for interpreting the deep learning model. From the interpretation, radiologists have some understanding of how the deep learning model makes the predictions, while data scientists can find the existing problems in the deep learning model to improve its performance.

Although having some limitations, the *deoxys* framework still has the potential of improvement and extension. This includes implementing advanced techniques into the existing visualization methods and adding other model visualization methods into the framework. Ideally, to utilize all modules implemented in the *deoxys* framework, an interactive user interface should be developed so that radiologists and data scientists can use the *deoxys* framework effectively and effortlessly.

Bibliography

[1] World Health Organisation, *All cancers fact sheet*, Retrieved 2019-11-20, from http://gco.iarc.fr/today/data/factsheets/cancers/39-All-cancers-fact-sheet.pdf, 2018.

[2] S. Gudi, S. Ghosh-Laskar, J. P. Agarwal, S. Chaudhari, V. Rangarajan, S. Nojin Paul, R. Upreti, V. Murthy, A. Budrukkar and T. Gupta, 'Interobserver variability in the delineation of gross tumour volume and specified organs-at-risk during imrt for head and neck cancers and the impact of fdg-pet/ct on such variability at the primary site', *Journal of Medical Imaging and Radiation Sciences*, vol. 48, no. 2, pp. 184–192, Jun. 2017, ISSN: 1939-8654. DOI: 10.1016/j.jmir.2016.11.003. [Online]. Available: https://doi.org/10.1016/j.jmir.2016.11.003.

[3] C. Njeh, 'Tumor delineation: The weakest link in the search for accuracy in radiotherapy', *Journal of Medical Physics*, vol. 33, no. 4, pp. 136–140, 2008. DOI: 10.4103/0971-6203.44472.

[4] E. Weiss and C. F. Hess, 'The impact of gross tumor volume (gtv) and clinical target volume (ctv) definition on the total accuracy in radiotherapy', *Strahlentherapie und Onkologie*, vol. 179, no. 1, pp. 21–30, Jan. 2003, ISSN: 0179-7158. DOI: 10.1007/s00066-003-0976-5. [Online]. Available: https://doi.org/10.1007/s00066-003-0976-5.

[5] E. Rusten, B. L. Rekstad, C. Undseth, G. Al-Haidari, B. Hanekamp, E. Hernes, T. P. Hellebust, E. Malinen and M. G. Guren, 'Target volume delineation of anal cancer based on magnetic resonance imaging or positron emission tomography', *Radiation Oncology*, vol. 12, no. 1, p. 147, 2017, ISSN: 1748-717X. DOI: 10.1186/s13014-017-0883-z. [Online]. Available: https://doi.org/10.1186/s13014-017-0883-z.

[6] P. M. Harari, S. Song and W. A. Tomé, 'Emphasizing conformal avoidance versus target definition for imrt planning in head-and-neck cancer', *International journal of radiation oncology, biology, physics*, vol. 77, no. 3, pp. 950–

958, Jul. 2010, ISSN: 1879-355X. DOI: 10.1016/j.ijrobp.2009.09.062. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/20378266.

[7] W. H. Organization *et al.*, 'Density of physicians (total number per 1000 population, latest available year)', *Global Health Observatory (GHO) data*, 2017.

[8] B. H. Kann, R. Thompson, C. R. J. Thomas, A. Dicker and S. Aneja, 'Artificial intelligence in oncology: Current applications and future directions', *Oncology (Williston Park, N.Y.)*, vol. 33, no. 2, pp. 46–53, Feb. 2019, 622430[PII], ISSN: 0890-9091. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30784028.

[9] S. Li, J. Xiao, L. He, X. Peng and X. Yuan, 'The tumor target segmentation of nasopharyngeal cancer in ct images based on deep learning methods', *Technology in Cancer Research & Treatment*, vol. 18, p. 1 533 033 819 884 561, 2019, PMID: 31736433. DOI: 10.1177/1533033819884561. eprint: https://doi.org/10.1177/1533033819884561. [Online]. Available: https://doi.org/10.1177/1533033819884561.

[10] B. Huang, Z. Chen, P.-M. Wu, Y. Ye, S.-T. Feng, C.-Y. O. Wong, L. Zheng, Y. Liu, T. Wang, Q. Li and B. Huang, 'Fully automated delineation of gross tumor volume for head and neck cancer on pet-ct using deep learning: A dual-center study', eng, *Contrast media & molecular imaging*, vol. 2018, p. 8 923 028, 2018, ISSN: 1555-4309.

[11] Y. M. Moe, 'Deep learning for automatic delineation of tumours from PET/CT images', Master's thesis, NMBU, 2019.

[12] S. Trebeschi, J. van Griethuysen, D. Lambregts, M. Lahaye, C. Parmar, F. Bakers, N. Peters, R. Beets-Tan and H. Aerts, 'Deep learning for fully-automated localization and segmentation of rectal cancer on multiparametric mr', eng, *Sci Rep*, vol. 7, no. 1, pp. 5301–5301, 2017, ISSN: 2045-2322.

[13] T. Lustberg, J. van Soest, M. Gooding, D. Peressutti, P. Aljabar, J. van der Stoep, W. van Elmpt and A. Dekker, 'Clinical evaluation of atlas and deep learning based automatic contouring for lung cancer', *Radiotherapy and Oncology*, vol. 126, no. 2, pp. 312–317, 2018.

[14] C. K. Kaushal, *Deep learning for automatic tumor delineation of anal cancer based on mri, pet and ct images*, eng, 2019. [Online]. Available: http://hdl.handle.net/11250/2605613.

[15] D. Erhan, Y. Bengio, A. Courville and P. Vincent, 'Visualizing higher-layer features of a deep network', *Technical Report, Univeristé de Montréal*, Jan. 2009.

[16] A. Mahendran and A. Vedaldi, 'Understanding deep image representations by inverting them', in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: 10.1109/cvpr.2015.7299155. [Online]. Available: https://doi.org/10.1109/cvpr.2015.7299155.

[17] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy and J. Yosinski, 'Plug &amp; Play Generative Networks: Conditional Iterative Generation of Images in Latent Space', *arXiv e-prints*, arXiv:1612.00005, arXiv:1612.00005, Nov. 2016. arXiv: 1612.00005 [cs.CV].

[18] K. Simonyan, A. Vedaldi and A. Zisserman, 'Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps', *arXiv e-prints*, arXiv:1312.6034, arXiv:1312.6034, Dec. 2013. arXiv: 1312.6034 [cs.CV].

[19] M. D. Zeiler and R. Fergus, 'Visualizing and Understanding Convolutional Networks', *arXiv e-prints*, arXiv:1311.2901, arXiv:1311.2901, Nov. 2013. arXiv: 1311.2901 [cs.CV].

[20] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. Riedmiller, 'Striving for Simplicity: The All Convolutional Net', *arXiv e-prints*, arXiv:1412.6806, arXiv:1412.6806, Dec. 2014. arXiv: 1412.6806 [cs.LG].

[21] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva and A. Torralba, 'Learning Deep Features for Discriminative Localization', *arXiv e-prints*, arXiv:1512.04150, arXiv:1512.04150, Dec. 2015. arXiv: 1512.04150 [cs.CV].

[22] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh and D. Batra, 'Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization', *arXiv e-prints*, arXiv:1610.02391, arXiv:1610.02391, Oct. 2016. arXiv: 1610.02391 [cs.CV].

[23] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[24] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017, ISBN: 9781617294433.

[25] S. Raschka and V. Mirjalili, *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow 2, 3rd Edition*. Packt Publishing, 2019, ISBN: 9781789955750. [Online]. Available: https://books.google.no/books?id=n1cjyAEACAAJ.

[26] Ö. F. Ertuğrul, 'A novel type of activation function in artificial neural networks: Trained activation function', *Neural Networks*, vol. 99, pp. 148–157, Mar. 2018. DOI: 10.1016/j.neunet.2018.01.007. [Online]. Available: https://doi.org/10.1016/j.neunet.2018.01.007.

[27] Y. Wang, Y. Li, Y. Song and X. Rong, 'The influence of the activation function in a convolution neural network model of facial expression recognition', *Applied Sciences*, vol. 10, no. 5, p. 1897, Mar. 2020. DOI: `10.3390/app10051897`. [Online]. Available: `https://doi.org/10.3390/app10051897`.

[28] H. Chieng, N. Wahid, O. Pauline and S. R. K. Perla, 'Flatten-t swish: A thresholded relu-swish-like activation function for deep learning', *International Journal of Advances in Intelligent Informatics*, vol. 4, no. 2, pp. 76–86, 2018, ISSN: 2548-3161. DOI: `10.26555/ijain.v4i2.249`. [Online]. Available: `http://ijain.org/index.php/IJAIN/article/view/249`.

[29] X. Glorot, A. Bordes and Y. Bengio, 'Deep sparse rectifier neural networks', in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[30] A. Shrikumar, P. Greenside, A. Shcherbina and A. Kundaje, *Not just a black box: Learning important features through propagating activation differences*, 2016. arXiv: `1605.01713 [cs.LG]`.

[31] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[32] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[33] K. Janocha and W. M. Czarnecki, 'On Loss Functions for Deep Neural Networks in Classification', *arXiv e-prints*, arXiv:1702.05659, arXiv:1702.05659, Feb. 2017. arXiv: `1702.05659 [cs.LG]`.

[34] R. Yu and L. Shi, 'A user-based taxonomy for deep learning visualization', *Visual Informatics*, vol. 2, no. 3, pp. 147–154, Sep. 2018. DOI: `10.1016/j.visinf.2018.09.001`. [Online]. Available: `https://doi.org/10.1016/j.visinf.2018.09.001`.

[35] W. Burger and M. J. Burge, *Digital Image Processing*. Springer London, 2016. DOI: `10.1007/978-1-4471-6684-9`. [Online]. Available: `https://doi.org/10.1007/978-1-4471-6684-9`.

[36] V. Dumoulin and F. Visin, *A guide to convolution arithmetic for deep learning*, 2016. arXiv: `1603.07285 [stat.ML]`.

[37] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. arXiv: `1409.1556 [cs.CV]`.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, 'Imagenet: A large-scale hierarchical image database', in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[39] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[40] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P.-L. Carrier, K. Cho, J. Chorowski, P. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrancois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. T. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, É. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang and Y. Zhang, *Theano: A python framework for fast computation of mathematical expressions*, 2016. arXiv: `1605.02688 [cs.SC]`.

[41] J. Rodríguez, A. Pérez and J. Lozano, ‘Sensitivity analysis of k-fold cross validation in prediction error estimation’, *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, pp. 569–575, Apr. 2010.

[42] O. Ronneberger, P. Fischer and T. Brox, ‘U-net: Convolutional networks for biomedical image segmentation’, N. Navab, J. Hornegger, W. M. Wells and A. F. Frangi, Eds., pp. 234–241, 2015.

[43] H. Abbes and F. Gargouri, 'MongoDB-based modular ontology building for big data integration', *Journal on Data Semantics*, vol. 7, no. 1, pp. 1–27, Oct. 2017. DOI: `10.1007/s13740-017-0081-z`. [Online]. Available: `https://doi.org/10.1007/s13740-017-0081-z`.

[44] D. A. Pereira, W. O. de Morais and E. P. de Freitas, 'NoSQL real-time database performance comparison', *International Journal of Parallel, Emergent and Distributed Systems*, vol. 33, no. 2, pp. 144–156, Mar. 2017. DOI: `10.1080/17445760.2017.1307367`. [Online]. Available: `https://doi.org/10.1080/17445760.2017.1307367`.

[45] O. Ronneberger, P. Fischer and T. Brox, 'U-net: Convolutional networks for biomedical image segmentation', *CoRR*, vol. abs/1505.04597, 2015. arXiv: `1505.04597`. [Online]. Available: `http://arxiv.org/abs/1505.04597`.

[46] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: `1412.6980 [cs.LG]`.

[47] L. R. Dice, 'Measures of the amount of ecologic association between species', *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[48] A. E. Hoerl and R. W. Kennard, 'Ridge regression: Biased estimation for nonorthogonal problems', *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[49] G. N. Hounsfield, 'Computed medical imaging', *Journal of Computer Assisted Tomography*, vol. 4, no. 5, pp. 665–674, Oct. 1980. DOI: `10.1097/00004728-198010000-00017`. [Online]. Available: `https://doi.org/10.1097/00004728-198010000-00017`.

[50] G. Lucignani, G. Paganelli and E. Bombardieri, 'The use of standardized uptake values for assessing FDG uptake with PET in oncology: A clinical perspective', *Nuclear Medicine Communications*, vol. 25, no. 7, pp. 651–656, Jul. 2004. DOI: `10.1097/01.mnm.0000134329.30912.49`. [Online]. Available: `https://doi.org/10.1097/01.mnm.0000134329.30912.49`.

[51] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert and B. Glocker, 'Efficient multi-scale 3d CNN with fully connected CRF for accurate brain lesion segmentation', *Medical Image Analysis*, vol. 36, pp. 61–78, Feb. 2017. DOI: `10.1016/j.media.2016.10.004`. [Online]. Available: `https://doi.org/10.1016/j.media.2016.10.004`.

[52] L. Lin, Q. Dou, Y.-M. Jin, G.-Q. Zhou, Y.-Q. Tang, W.-L. Chen, B.-A. Su, F. Liu, C.-J. Tao, N. Jiang, J.-Y. Li, L.-L. Tang, C.-M. Xie, S.-M. Huang, J. Ma, P.-A. Heng, J. T. S. Wee, M. L. K. Chua, H. Chen and Y. Sun, 'Deep learning for automated contouring of primary tumor volumes by MRI for nasopharyngeal carcinoma', *Radiology*, vol. 291, no. 3, pp. 677–686, Jun.

2019. DOI: 10.1148/radiol.2019182012. [Online]. Available: https://doi.org/10.1148/radiol.2019182012.

[53] K. Men, X. Chen, Y. Zhang, T. Zhang, J. Dai, J. Yi and Y. Li, ‘Deep deconvolutional neural network for target segmentation of nasopharyngeal cancer in planning computed tomography images’, *Frontiers in Oncology*, vol. 7, Dec. 2017. DOI: 10.3389/fonc.2017.00315. [Online]. Available: https://doi.org/10.3389/fonc.2017.00315.

[54] C. E. Cardenas, B. M. Anderson, M. Aristophanous, J. Yang, D. J. Rhee, R. E. McCarroll, A. S. R. Mohamed, M. Kamal, B. A. Elgohari, H. M. Elhalawani, C. D. Fuller, A. Rao, A. S. Garden and L. E. Court, ‘Auto-delineation of oropharyngeal clinical target volumes using 3d convolutional neural networks’, *Physics in Medicine & Biology*, vol. 63, no. 21, p. 215 026, Nov. 2018. DOI: 10.1088/1361-6560/aae8a9. [Online]. Available: https://doi.org/10.1088/1361-6560/aae8a9.

[55] Z. Guo, N. Guo, K. Gong, S. Zhong and Q. Li, ‘Gross tumor volume segmentation for head and neck cancer radiotherapy using deep dense multi-modality network’, *Physics in Medicine & Biology*, vol. 64, no. 20, p. 205 015, Oct. 2019. DOI: 10.1088/1361-6560/ab440d. [Online]. Available: https://doi.org/10.1088/1361-6560/ab440d.

[56] H. Chen, Q. Dou, L. Yu and P.-A. Heng, *Voxresnet: Deep voxelwise residual networks for volumetric brain segmentation*, 2016. arXiv: 1608.05895 [cs.CV].

[57] P. Natekar, A. Kori and G. Krishnamurthi, ‘Demystifying brain tumor segmentation networks: Interpretability and uncertainty analysis’, *Frontiers in Computational Neuroscience*, vol. 14, Feb. 2020. DOI: 10.3389/fncom.2020.00006. [Online]. Available: https://doi.org/10.3389/fncom.2020.00006.

[58] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012. arXiv: 1207.0580 [cs.NE].

[59] M. A. Tanner and W. H. Wong, ‘The calculation of posterior distributions by data augmentation’, *Journal of the American statistical Association*, vol. 82, no. 398, pp. 528–540, 1987.

[60] L. Perez and J. Wang, *The effectiveness of data augmentation in image classification using deep learning*, 2017. arXiv: 1712.04621 [cs.CV].

[61] Z. Hussain, F. Gimenez, D. Yi and D. Rubin, ‘Differential data augmentation techniques for medical imaging classification tasks’, in *AMIA Annual Symposium Proceedings*, American Medical Informatics Association, vol. 2017, 2017, p. 979.

[62] K. Wickstrøm, M. Kampffmeyer and R. Jenssen, 'Uncertainty and interpretability in convolutional neural networks for semantic segmentation of colorectal polyps', *Medical Image Analysis*, vol. 60, p. 101 619, Feb. 2020. DOI: `10.1016/j.media.2019.101619`. [Online]. Available: `https://doi.org/10.1016/j.media.2019.101619`.

[63] X. Zheng, R. Ji, X. Sun, W. Yongjian, F. Huang and Y. Yang, 'Centralized ranking loss with weakly supervised localization for fine-grained object retrieval', Jul. 2018, pp. 1226–1233. DOI: `10.24963/ijcai.2018/171`.

[64] C. Etmann, S. Lunz, P. Maass and C.-B. Schönlieb, *On the connection between adversarial robustness and saliency map interpretability*, 2019. arXiv: `1905.04172 [stat.ML]`.

[65] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner and A. Madry, 'Robustness may be at odds with accuracy', in *International Conference on Learning Representations*, 2019. [Online]. Available: `https://openreview.net/forum?id=SyxAb30cY7`.

[66] A. Diamant, A. Chatterjee, M. Vallières, G. Shenouda and J. Seuntjens, 'Deep learning in head & neck cancer outcome prediction', *Scientific Reports*, vol. 9, no. 1, Feb. 2019. DOI: `10.1038/s41598-019-39206-1`. [Online]. Available: `https://doi.org/10.1038/s41598-019-39206-1`.

[67] *Tensorflow 2.0 is now available!*, Sep. 2019. [Online]. Available: `https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html`.

[68] A. Mordvintsev, C. Olah and M. Tyka, 'Inceptionism: Going deeper into neural networks', 2015. [Online]. Available: `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`.

[69] D. Strong and T. Chan, 'Edge-preserving and scale-dependent properties of total variation regularization', *Inverse problems*, vol. 19, no. 6, S165, 2003.

[70] A. L. Maas, A. Y. Hannun and A. Y. Ng, 'Rectifier nonlinearities improve neural network acoustic models'.

[71] B. Xu, N. Wang, T. Chen and M. Li, *Empirical evaluation of rectified activations in convolutional network*, 2015. arXiv: `1505.00853 [cs.LG]`.

Appendix A

DAT390 Data Science Seminar report

This is the report for the coursework DAT390 Data Science Seminar in NMBU[1]. This report describes the Software Requirement Specification and Software Design Document of the *deoxys* framework, which was a preparation of this Master's thesis.

---

[1] The course information is available at `https://www.nmbu.no/course/DAT390?studieaar=2019`.

Development of a Keras-based CNN framework for automatic delineation of cancer tumors

Bao Ngoc Huynh

Nov 21st 2019. Last modified on May 20th 2020.

Abstract

This is the report for the project in DAT390 course in NMBU. This report provides the results of the development of Keras-based framework for automatic tumor delineation. It contains the Software Requirement Specification, as well as the Design Document for the resulting framework. A resulting framework has been successfully developed with the minimum requirement to run an experiment after configuring a convolutional neural network. The neural network created from the experiment can automatically delineate cancer tumors from medical images. The delineation can be used as an external opinion to help radiologists in the process of radiotherapy for cancer treatment.

1 Introduction

Cancer is a deadly disease, which is responsible for over nine million death in 2018 [9]. Therefore, it is crucial to find effective and efficient treatments. One of the most effective cancer treatments is radiotherapy, where cancer cells are killed using doses of radiation. However, the irradiation process not only kills cancer tumors but also affects healthy tissues surrounding the cancer tumor. Thus, accuracy in radiotherapy has to be increased to minimize the radiation dose delivered to healthy cells and maximize the dose to cancer tumors. If all of the radiotherapy steps are linked in a chain, tumor delineation is the weakest link, and its accuracy significantly impacts radiotherapy accuracy [2][5]. Therefore, increasing the accuracy of tumor delineation is one of the challenges of radiotherapy treatment. Furthermore, a study conducted by Weiss and Hess [8] shows that due to interobserver variability, when different radiologists delineate the same case, the variation of gravity centers of these tumors is up to 0.6-0.7cm. When analyzing the result of interobserver, the uncertainty in delineation is even larger than patient positioning and organ motion [7][8]. Therefore, one method of improving the radiotherapy accuracy is having more than one radiologist in one case. However, the long waiting time to delineate tumor (4 hours for a trained radiologist) [3] makes this method almost impossible. However, with the increasing of technology, tumor delineation can now be done automatically using deep learning, to be specific, using a convolutional neural network (CNN). The automatic delineation results can be used as one “observer” in radiotherapy.

In order to delineate tumor like a radiologist, a deep learning model has to be created. This process contains repeating a group of actions such as model training (learning from radiologists), model testing, and model modification until the best model, which performs most similar to the radiologist, is found. The author calls this group of actions an experiment. Currently, there are many software libraries that help to create a deep learning model. However, these libraries are not specialized for CNN in medical images. Besides, creating and running experiments have to be done manually, which consumes time and effort.

The goal of this project in DAT390 course is to create a Software Requirement Specification and a Software Design to develop a framework which automatically delineates cancer tumor as well as resolve the disadvantage of the existing deep learning libraries. The developed framework should satisfy the minimum requirements, in which users can use the resulted framework to perform a single experiment to delineate cancer tumors automatically.

This report will define a software requirement specification (SRS) and Design Document of a Keras-based framework for the automatic delineation of cancer tumors. Moreover, this report also includes the progress of the development based on the SRS and Design and the current results.

2 Theory and Definition

2.1 Convolutional Neural Network

Artificial Intelligence (AI) refers to the term of giving the machines human "brain" so that it can "think" as well as "act" like a human. Machine learning (ML), a subfield of AI, focuses on making predictions based on existing data. Deep learning is one of an approach in machine learning where the data is learned through layers of a neural network. In the context of automatic tumour delineation, AI means making a computer program to perform radiologists' jobs. In contrast, ML means analyzing each pixel in medical images and deciding if it belongs to a cancer tumor. Deep learning refers to one of the approaches of predicting cancer tumors by transforming the image data through a number of layers in a neural network.

Figure 1 illustrates how a neural network learns. If we apply this figure to the current context, *Input* X refers to the set of medical images to be delineated. *True targets* Y refers to the delineation by a real radiologist. *Predictions* Y' refers to the delineation made by the neural network. The images data (*Input* X) go through each *layer* and transform by applying some *weights* in that layer. The outputs of each layer may be used as the inputs of another layer. This creates a network of layers. In many cases, an activation function is applied to the output of the layer. The output of the final layers is the *Predictions* Y'. By using the *Loss score* calculated by applying a *Loss function* on *Predictions* Y' and *True targets* Y, the *Optimizer* updates the *weights*. The goal of this learning process is to find the weights that minimize the *Loss score* so that the neural network can make predictions closest to the radiologist.

Convolutional Neural Network (CNN) is the type of neural network containing layers using a convolution filter to transform the images data. There are many types of convolution filters with different effects on the image data, either dilation, erosion, shrinkage, expansion, etc.

2.2 Sequential Architecture

Architecture refers to the structure of the layers in the CNN. It determines how the layers are connected in the neural network. In the CNN with sequential architecture, the layers are connected sequentially. That means there are only one input and one output for each layer, and the output of the preceding layer is the input of the next layer. We can imagine the sequential CNN as a stack of layers, where the input will penetrate through all these layers in a straight line.

2.3 U-Net Architecture

In the U-net Architecture, the image data go through two paths: the downsampling (contraction) path and the upsampling (expansion) path. In the original paper, the U-net architecture is defined in figure 2

2.4 Deep Learning Experiment

Experiments of deep learning to find a good enough model for automatic delineation should follow these steps:

Step 1. Choose the metrics. This is the criteria to determine if a CNN has been trained well enough to use as an "observer".

Step 2. Prepare the input data. The input data here means the medical images to be delineated. Typically, the data is split into three sets:

- Training data. This set of data is used for training the CNN.
- Validation data. This set is used for evaluating the performance of the CNN and tuning hyper-parameters.
- Test data. This set should be isolated from the other two sets. The performance the CNN is decided by evaluating this set, not the validation data set.

Step 3. Define architecture to use in the CNN.

Step 4. Choose hyper-parameters. The hyper-parameters are any mutable object in the CNN. This includes the loss function, optimizer, the type of convolution layer, the activation, etc.

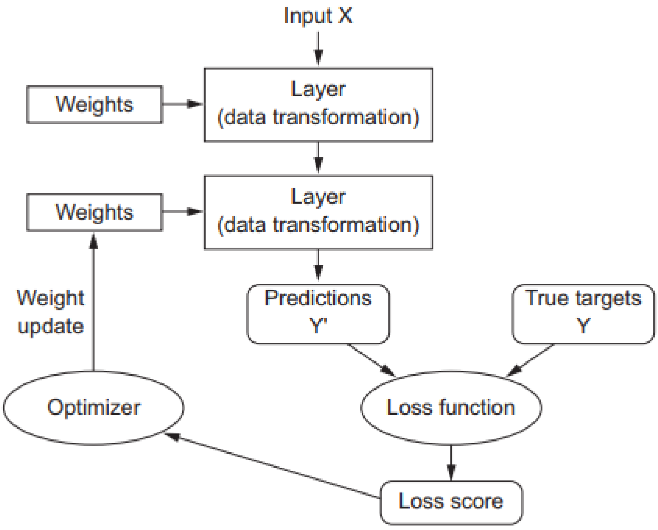


Figure 1: The Learning Process of a Neural Network. The Input X goes through some data transformation layers by applying some weights in each layers. After going through all layers, the final transformed data, called Predictions Y', combines with the True targets Y in a loss function to calculate the loss score. The optimizer uses this score to update the weights until the neural network finds the weights that minimises the Loss score. Orignated from F. Chollet. Deep learning with Python [1]

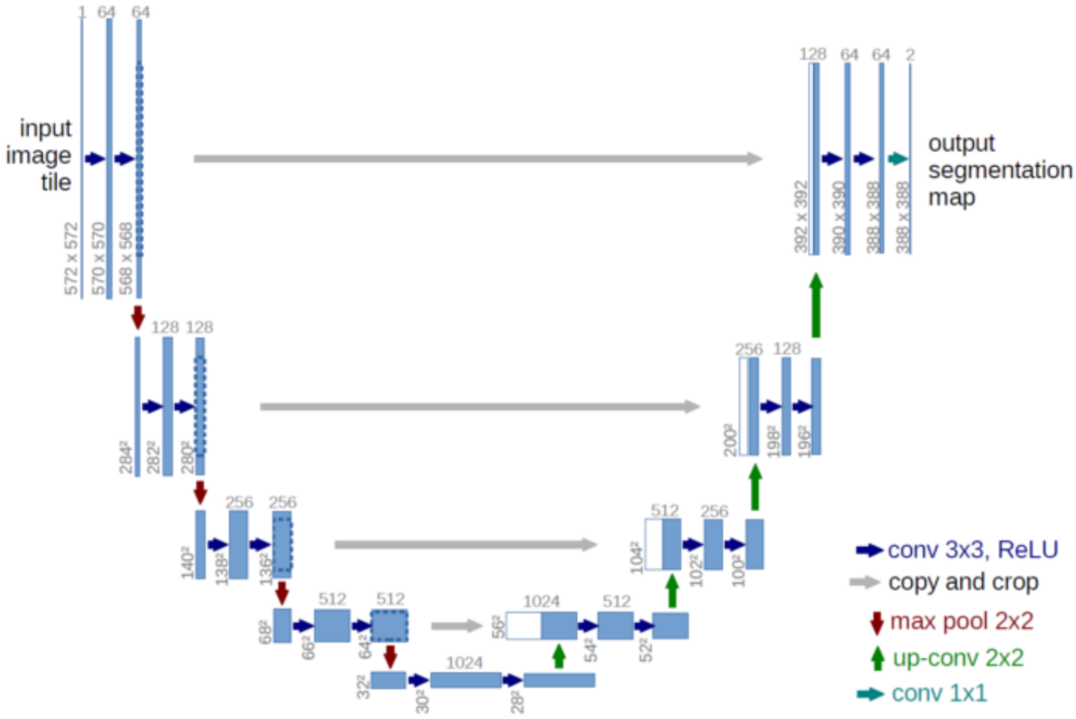


Figure 2: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations [6]. Originate from Olaf Ronneberger et al [6]

Step 5. Train the model on the training data with the selected hyper-parameters.

Step 6. Evaluate the performance of the model on the validation data with the metric chosen in step 1.

Step 7. Repeat step 4 until the combination of hyper-parameters with the chosen architecture that makes the best model based on the chosen metric is found.

Step 8. Repeat step 3 until an architecture, together with a combination of hyper-parameters that create the best model is found.

Step 9. Use the best model to check the performance on test data.

An experiment refers to the process from step 1 to step 6.

3 The Framework

The development of the resulted framework, named *deoxys*, has the goal of providing the users the ability to run multiple experiments of different CNN models and then choose the best model for final prediction. This framework should be specialized in deep-learning in medical images, especially in auto-delineation of cancer tumor. Because of that, it should integrate u-net architecture and image preprocessing modules, as well as logging tools and performance visualization tools when running experiments. These are the minimum requirements of the framework. It can be later extended with other types of architectures, preprocessors, automation, interactive verbose configuration, and visualization.

The development time, as well as maintenance time, will range from October 1st, 2019 until May 1st, 2020. The first milestone is on January 6th, 2020, with the goal of creating a framework that satisfies the minimum requirements, which will be defined in detail in the software requirement specification (see 3.1).

3.1 Software Requirement Specification

This part defines the requirement specification of the developing framework. Because of that, terms indicating future-tense such as “should”, “shall”, “will” as well as terms indicating ability such as “can”, “may” will be used when describing framework.

In order to reach the goal of the development, the *deoxys* framework should satisfy all the requirements defined in User Requirement Specification (see 3.1.1) and System Requirement Specification (see 3.1.3)

3.1.1 User Requirement Specification

Users are referring to master students, Ph.D. candidates, researchers, and anyone who wants to use deep-learning on the automatic delineation of cancer tumors. This framework is targeted to the users with basic programming knowledge, including the usage of JSON data structure, and with the knowledge of deep learning, especially in convolutional neural network. Basic programming knowledge is including but not limited to object-oriented programming in python, other python libraries such as matplotlib, Keras, h5py.

With the help of *deoxys*, users shall have the ability to perform multiple CNN experiments by creating configurable JSON files. Users can define their own sequential or u-net model with the choices of layers, loss functions, optimizers, metrics and many other hyper-parameters. In addition, users can choose how to split the data for training, validation and testing. Each experiment should include training the data, logging the performance and evaluation of the trained model on test data. All trained models can be saved to disk and loaded back for the continuation of training or any other purposes.

As a follow-up after running an experiment, users can also check the predicted outputs as delineated images in comparison with the original image and view the performance graphs of the trained model.

Users with advanced programming knowledge can also customize and create their custom model architecture, layers, activation functions, loss functions, optimizers, metrics, etc…

3.1.2 Use cases

From the user requirement specification, the *deoxys* framework should support the following six use cases:

1. Create a model
2. Train a model
3. Save a trained model
4. Load a model from file(s)

5. Set up an experiment
6. Create and apply customized model objects to the model

Use case diagram Figure 3 shows all the use cases and their interaction inside the framework. There are three main flows of the use cases:

- Setting up an experiment using configurations to run and evaluate that experiment. This starts with creating a model from the configuration, then setting up an experiment by training and evaluating the configured model.
- Loading and saving trained model from and to disk.
- Creating customize objects / elements for the experiment. This includes: Layers, Activation functions, Loss functions, Optimizers, Callbacks.

Use case 1: Create a model Every action of the user involves the use of the model. The model term in the *deoxys* framework refers to a group of three components. The first component is a convolutional neural network, which can be a sequential CNN or an U-net CNN, or even a customized CNN defined by the users. This CNN contains input shapes, layers, activation functions. We call this component the architecture of the model. The second component is the set of hyper-parameters of the neural network, which includes the optimizers, loss function, and metrics. The last component, called Data Reader, acts as a data provider, which feeds the data, medical images with delineation contour, into the neural network for training and evaluation. This involves splitting up the data into training data, validation data, and test data, as well as preprocessing the data to make sure the data is suitable for training in the CNN.

Use case 2: Train a model Since a model contains all the components needed for training, the training process can be performed directly after a model is created.

Use case 3: Save a trained model After the model is trained, it is necessary to save that model for later use, either to bring the saved model to another location to perform automatic tumor delineation, or to continue training.

Use case 4: Load a model from file(s) Because a model can be saved to files, users should be able to load it back to use it later.

Use case 5: Set up an experiment Instead of training a model directly, users can set up an experiment from a created model, or a saved model. This includes training the model while logging the performance of the models on the training data and the validation data using the predefined metrics. Besides, users can configure checkpoints for saving models and making predictions on validation data while training. Visualization of performance and predictions on test data can also be performed when after running an experiment.

Use case 6: Create and apply customized model objects to the model Model objects refer to the layers in the neural network, the activation functions, the optimizers, the loss functions, metrics, and any other components existing in a model. Since not all types of objects can be predefined, users with advanced programming knowledge should be able to define customized objects and apply it to their models.

3.1.3 System Requirement Specification

The *deoxys* framework should have the following attributes: usability, reliability, flexibility, maintainability and portability.

Usability The *deoxys* framework should be easy to install, learn and use. The expected training and learning time for a user to use this framework effectively should not take more than 40 hours. For this reason, this framework should have detailed documentation of the installation guide and usage of each class, function and property. It should also provide sample code snippets which can be applied to the defined use cases.

Reliability The output generated when running code from *deoxys* framework should have the behaviors as documented. In addition, the unexpected error rate should be under 5% and at least 80% of code lines should have been tested before release.

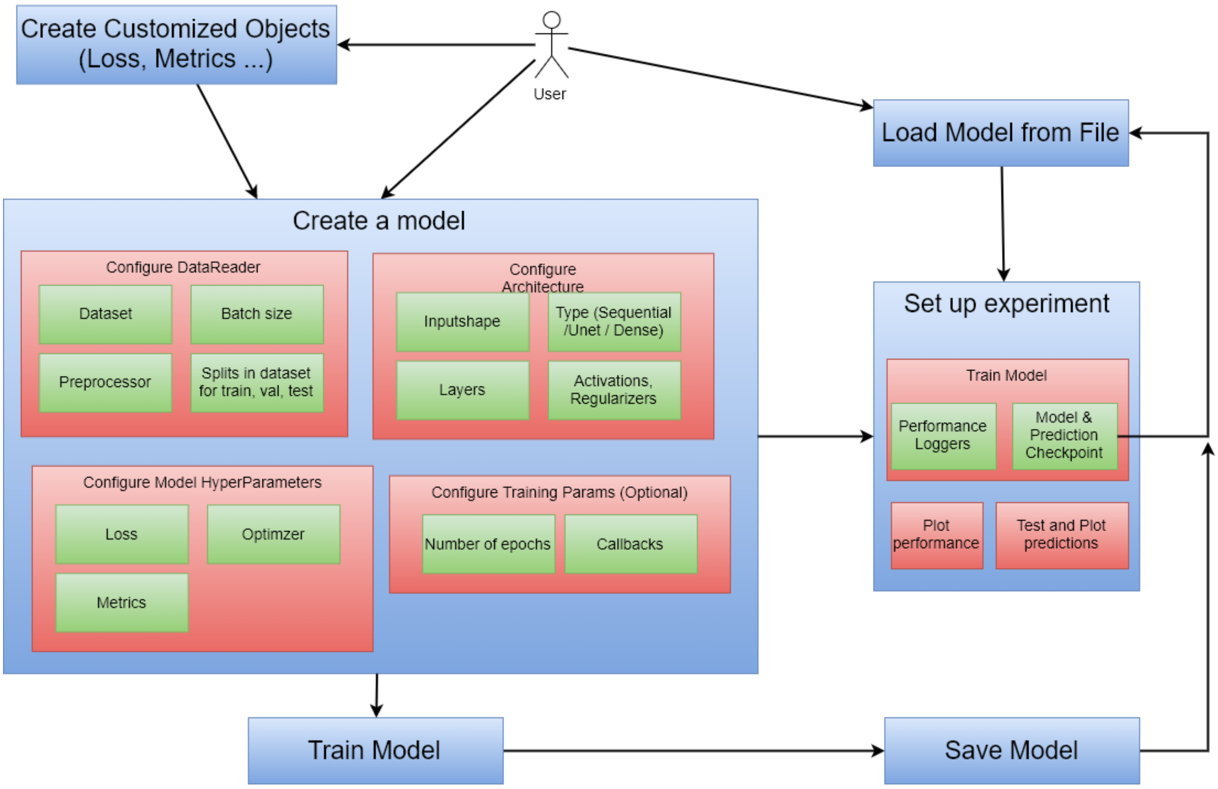

Create Customized Objects (Loss, Metrics ...)
User
Load Model from File
Create a model
Configure DataReader
Dataset
Batch size
Preprocessor
Splits in dataset for train, val, test
Configure Architecture
Inputshape
Type (Sequential /Unet / Dense)
Layers
Activations, Regularizers
Configure Model HyperParameters
Loss
Optimzer
Metrics
Configure Training Params (Optional)
Number of epochs
Callbacks
Set up experiment
Train Model
Performance Loggers
Model & Prediction Checkpoint
Plot performance
Test and Plot predictions
Train Model
Save Model

Figure 3: Use Case Diagram

Flexibility Users should be able to customize and create new components to integrate with *deoxys* framework.

Maintainability The *deoxys* framework should be easy to maintain. Therefore, it should be divided into separated modules. Moreover, all of the source code should follow the PEP8 coding convention. Also, this framework should log all actions in different versions and issues from the users.

Maintaining the framework includes fixing bugs, handling issues, updating and adding new features. The maintenance activities should last at least until May 2020.

Portability The *deoxys* framework should work properly when the following hardware requirements and environment are satisfied:

- System memory: at least 8GB with GPU or 13GB without GPU
- Python version: at least 3.7

3.2 Designs

3.2.1 Overview

Before development, the designs of the framework have to be considered.

The first things to concern are the usability and maintainability of the framework. As stated, in the previous sessions, all source code shall follow PEP8 coding convention. Sphinx will be used as the tool of documentation. In addition, git is used as a tool to handle logging and version management. All source code should be available in `http://github.com/huynhngoc/deoxys`.

Implementation of all layers and other components in convolutional neural networks within a three-month time is impossible. Therefore, Keras is used as a based library, as it contains implemented layers, activation functions, optimizers and other components in CNNs. Also, Keras is compatible with TensorFlow 1.x, 2.x, which is a powerful backend tool in deep-learning, as well as other backends such as Theano, etc.

The author suggests that the framework should have the following modules:

- Models: contains a wrapper of a Keras model. Other Keras objects such as optimizers, activation functions, etc are also included.
- Architecture loader. The loader should be able to create models from configurable JSON objects that contain the architecture of the model.
- Data reader: Since the target of this framework are medical images, the input data often has a large size and usually cannot fit into the computer memory. In order to avoid out of memory errors, this module should contain a data generator that split image data into smaller batches that can fit into the memory when training the model.
- Experiment: The *deoxys* framework should be able to perform a single experiment and multiple experiments.

Structure diagram Figure 4 illustrates the structure of the *deoxys* framework.

3.2.2 Model Objects

These modules are the components creating a model. They are layers, loss functions, activation functions, metrics, optimizer and callbacks. Any customized objects created by the users will be added to this module at runtime.

3.2.3 Model

Firstly, this module should be a wrapper of a Keras model. As a result, it should have methods of the Keras model, such as:

- `load`: loading models
- `save`: save models to files
- `fit`: fit a model with data
- `predict`: predict the target
- `evaluate`: evaluate the performance of the current state of the model

Secondly, it should have a Data Reader (see 3.2.5) instance, which provided proper inputs for actions on the model.

Finally, by performing methods in Keras model using the inputs from the data reader, the model should have the following methods:

- `fit_train`: fit the training data
- `predict_val`: predict the validation data
- `predict_test`: predict the test data
- `evaluate_test`: evaluate the performance of the current state of the model on the test data

3.2.4 Architecture Loader

This module should have a function to create a model from one of the predefined architecture. The predefined architectures are the sequential and the U-net. In the future, dense model should be implemented to be used as a predefined architecture. This module should be able to load a configurable JSON file to create a Keras model based on the configuration.

3.2.5 Data Reader

The data reader module should provide input data for training and evaluating the model. The data reader should provide three sets of data: training data, validation data, test data. These three sets should be in the form of a python generator, which is wrapped into a Data Generator. Using a python generator is essential because medical image data usually has a large size, and may not be able to fit into the running environment's memory. Using a python generator will feed the model with a small part of the data and minimize the chance of getting out of memory error. The list of preprocessors to be applied to the data should be configurable.

HDF5 Data Reader `h5` or `hdf5` is a file format that has the ability to store large dataset with compression and hierarchy, as well as meta-data. The main components of an HDF5 file are groups and datasets, where datasets are pieces of data that is stored in file while groups are containers of datasets.

The *deoxys* framework should have a HDF5 Data Reader, which is a Data Reader that process data from a hdf5 file. As a result, it should provide the three datasets: train, validation and test. Also, since an HDF5 file can be split into groups, the HDF5 Data Reader should provide an aid for configuring which groups of data to be in the three basic sets. It should

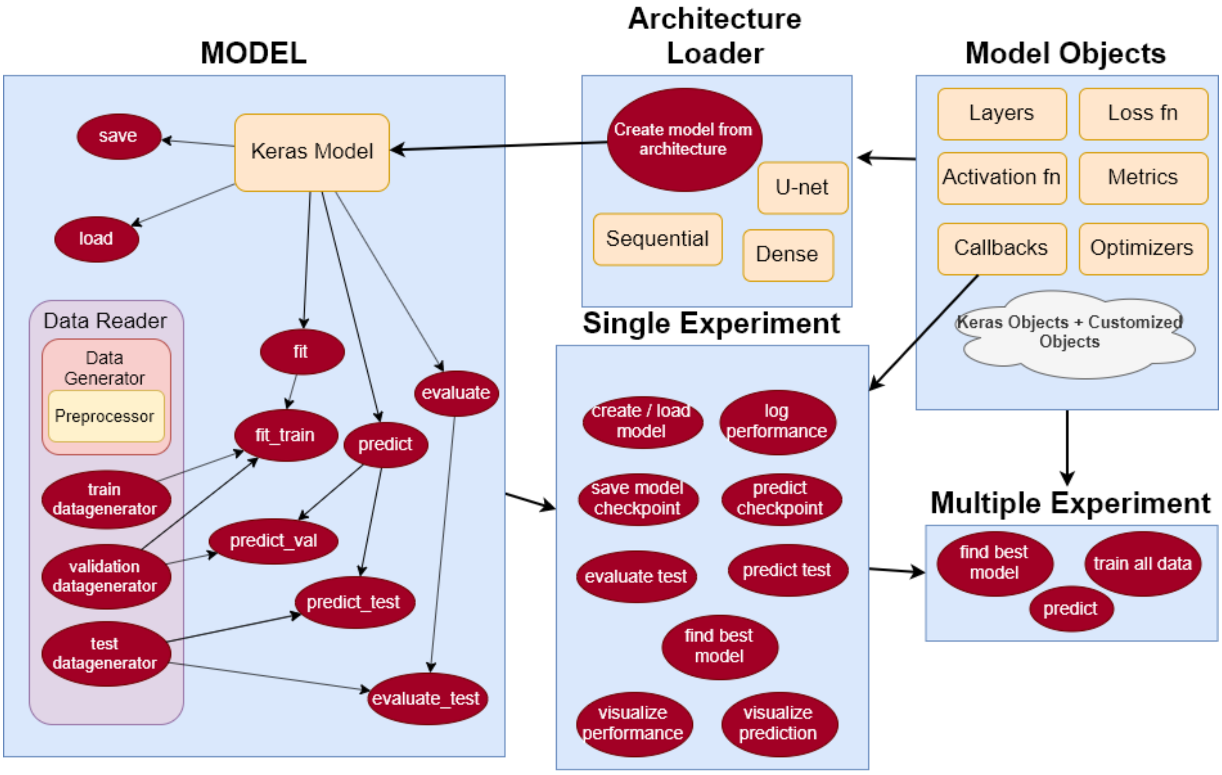


Figure 4: *Deoxys* Structure Diagram

be easy to configure different groups into different purposes for cross-validation. The suggested structure of hdf5 file to be used in the HDF5 Data Reader is to split the data into folds, where the users can configure which folds to be in the training set, or the validation set, or the test set.

Here is an example of the structure of a hdf5 file to be used in the HDF5 Data Reader, `\fold_[n]` is the name of the group, and `col_[n]` are names of the datasets, each of which is the column data.

```
\fold_0
   col_0
   col_1
   col_2
\fold_1
   col_0
   col_1
   col_2
\fold_2
   col_0
   col_1
   col_2
\fold_3
   col_0
   col_1
   col_2
\fold_4
   col_0
   col_1
   col_2
```

With this HDF5 file with the example structure, the HDF5 Data Reader should allow users to configure the following five things:

- Path to the HDF5 file
- The column to be used as *Input X*
- The column to be used as *True Target Y*
- The maximum number of items provided by the data generator. We call this number the `batch_size`.
- Which folds belong to which set. For example, users can configure `fold_0` and `fold_1` to be used for training, while `fold_2` is for validation and `fold_3` and `fold_4` are for testing. In another experiment, users can configure `fold_1` and `fold_2` to be used for training, while `fold_0` is for validation. In this way, users can use cross-validation in the frame-

work.

3.2.6 Experiment

Single Experiment The Single Experiment is used to perform an experiment. With the use of Keras 'callbacks', the modules can have the following actions while training:

- log training performance
- log validation performance
- save a model to disk at a checkpoint
- use the model at the checkpoint to predict validation data

By using the files created during training, the Single Experiment can visualize the training and validation performance, as well as visualize the predictions. The visualization of predictions can be either tumor delineation by radiologists and by the model directly on the original images, or a plot containing three images: the original images, the ground truth masks from radiologists, and the predictions of the model. Moreover, it can find the best model of each metric based on the log files.

Multiple Experiment The Multiple Experiment class should be able to run multiple single experiments, either concurrently or not. After finish all experiments, it should find the best model from all experiments and use that model to predict and evaluate performance on the test set.

4 Results

4.1 Implementation progress

4.1.1 Completed modules

By the time this report is submitted, users can perform a single experiment, with saving, loading, and visualization using the *deoxys* framework. This means all parts but "multiple experiments" from the design diagram in figure 4 have been implemented.

4.1.2 In-progress modules

Modules related to running multiple experiments are still in development. There are problems involving the process of combining multiple single experiments into a batch of experiments, as well as the concurrent programming that allows running multiple experiments in parallel.

Besides, there is still a lack of tests and documentation that needs to be resolved.

4.2 Run on test data

The data from Oslo University hospital was used for running a test experiment. It contained the CT and PET images to detect head and neck cancer. The model parameters were taken from Yngve Mardal Moe's master thesis [4] and run the training set with only 3000 slices of images and three epochs. A criterion of success is that the trained model has the performance of f-beta score above 0.5.

The result was amazing as the dice (f-beta score) was about 0.5 (figure 5) and some samples had nice delineation results (figure 6)

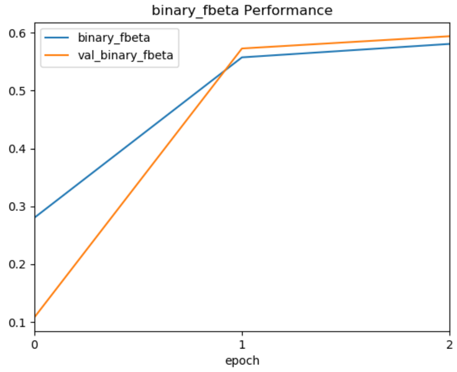


Figure 5: Visualization of performance (binary f-beta score) of an experiment

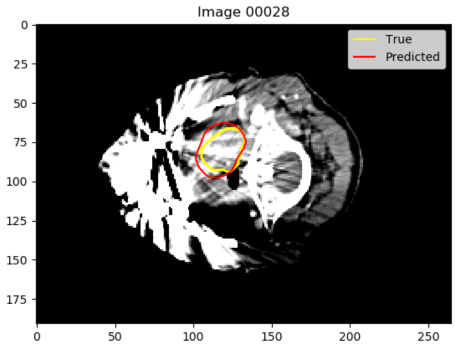


Figure 6: Visualization of predictions of a sample in an experiment

5 Discussions

As there are some meaningful results from the development, I can use this framework for running experiments in the Master's thesis. There is plenty of space for improvement. Possible improvements are:

- More customized preprocessors and callbacks should be added to the framework.
- Development of an auto-generated configuration tool, either web-based or verbose terminal tool.
- Back-propagation implementation based on the implemented model. This will be developed in my Master's thesis.
- Visualize the progress of training/prediction.
- Data generator as a sequential model for multiprocessing

6 Conclusion

By the time of creating this report, the development of the *deoxys* framework has satisfied the minimum requirements for running a single experiment. Users who are interested in automatic tumor delineation using deep-learning can try to create a model and run the experiment. The current development of *deoxys* can continue with many extensions and upgrades to create a more user-friendly framework with more features.

References

[1] François Chollet. *Deep Learning with Python*. Manning, November 2017. ISBN 9781617294433.

[2] Shivakumar Gudi, Sarbani Ghosh-Laskar, Jai Prakash Agarwal, Suresh Chaudhari, Venkatesh Rangarajan, Siji Nojin Paul, Rituraj Upreti, Vedang Murthy, Ashwini Budrukkar, and Tejpal Gupta. Interobserver variability in the delineation of gross tumour volume and specified organs-at-risk during imrt for head and neck cancers and the impact of fdg-pet/ct on such variability at the primary site. *Journal of Medical Imaging and Radiation Sciences*, 48(2):184–192, Jun 2017. ISSN 1939-8654. doi: 10.1016/j.jmir.2016.11.003. URL https://doi.org/10.1016/j.jmir.2016.11.003.

[3] Paul M. Harari, Shiyu Song, and Wolfgang A. Tomé. Emphasizing conformal avoidance versus target definition for imrt planning in head-and-neck cancer. *International journal of radiation oncology, biology, physics*, 77(3):950–958, Jul 2010. ISSN 1879-355X. doi: 10.1016/j.ijrobp.2009.09.062. URL https://www.ncbi.nlm.nih.gov/pubmed/20378266.

[4] Yngve Mardal Moe. Deep learning for automatic delineation of tumours from PET/CT images. Master's thesis, NMBU, 2019.

[5] C. Njeh. Tumor delineation: The weakest link in the search for accuracy in radiotherapy. *Journal of Medical Physics*, 33(4):136–140, 2008. doi: 10.4103/0971-6203.44472.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. pages 234–241, 2015.

[7] Espen Rusten, Bernt Louni Rekstad, Christine Undseth, Ghazwan Al-Haidari, Bettina Hanekamp, Eivor Hernes, Taran Paulsen Hellebust, Eirik Malinen, and Marianne Grønlie Guren. Target volume delineation of anal cancer based on magnetic resonance imaging or positron emission tomography. *Radiation Oncology*, 12(1):147, 2017. ISSN 1748-717X. doi: 10.1186/s13014-017-0883-z. URL https://doi.org/10.1186/s13014-017-0883-z.

[8] Elisabeth Weiss and Clemens F. Hess. The impact of gross tumor volume (gtv) and clinical

target volume (ctv) definition on the total accuracy in radiotherapy. *Strahlentherapie und Onkologie*, 179(1):21–30, Jan 2003. ISSN 0179-7158. doi: 10.1007/s00066-003-0976-5. URL https://doi.org/10.1007/s00066-003-0976-5.

[9] World Health Organisation. All cancers fact sheet, 2018. Retrieved 2019-11-20, from http://gco.iarc.fr/today/data/factsheets/cancers/39-All-cancers-fact-sheet.pdf.

Appendix B

Models trained on ImageNet dataset

In this Master's thesis, two models that trained on the ImageNet dataset were introduced. This section provides the detailed architectures of these two models.

The VGG16 model [37] contains 16 trainable layers with the structure described in Table B.1. In this table, the convolutional layer and max pooling layer were described in Section 2.2.3 on page 15. The Dense layers are simply the hidden layers in the neural network. The flatten layer transforms tensor data into one-dimensional data.

The model used by Springenberg *et al.* [20] is described in Table B.2. In this table, the Global Average Pooling layer has the same effect as the Flatten layer in the VGG16 model.

Layer's name	Layer's type	Filter size	Number of filters
input	Input	Not available	Not available
block1_conv1	Convolutional	3x3	64
block1_conv2	Convolutional	3x3	64
block1_pool	Max Pooling	Not available	Not available
block2_conv1	Convolutional	3x3	128
block2_conv2	Convolutional	3x3	128
block2_pool	Max Pooling	Not available	Not available
block3_conv1	Convolutional	3x3	256
block3_conv2	Convolutional	3x3	256
block3_conv2	Convolutional	3x3	256
block3_pool	Max Pooling	Not available	Not available
block4_conv1	Convolutional	3x3	512
block4_conv2	Convolutional	3x3	512
block4_conv2	Convolutional	3x3	512
block4_pool	Max Pooling	Not available	Not available
block5_conv1	Convolutional	3x3	1024
block5_conv2	Convolutional	3x3	1024
block5_conv2	Convolutional	3x3	1024
block5_pool	Max Pooling	Not available	Not available
flatten	Flatten	Not available	Not available
fc1	Dense	Not available	Not available
fc2	Dense	Not available	Not available
Prediction	Dense (with softmax)	Not available	Not available

Table B.1: The detailed structure of the VGG16 model [37].

Layer's name	Layer's type	Filter size	Number of filters
input	Input	Not available	Not available
conv1	Convolutional	11x11	96
conv2	Convolutional	1x1	96
conv3	Convolutional	3x3	96
conv4	Convolutional	5x5	256
conv5	Convolutional	1x1	256
conv6	Convolutional	3x3	256
conv7	Convolutional	3x3	384
conv8	Convolutional	1x1	384
conv9	Convolutional	3x3	384
conv10	Convolutional	3x3	1024
conv11	Convolutional	1x1	1024
conv12	Convolutional	1x1	1000
global_pool	Global average pooling	Not available	Not available
softmax	Activation function	Not available	Not available

Table B.2: The detailed structure of the model used by Springenberg *et al.* in [20].

Appendix C

Database test report

By using the materials defined in Table 3.1 on page 38, the manual tests for database integration were performed and reported in the following document.

TEST REPORT

RESULT	
Total Test Cases:	7
Passed:	7
Fail:	0

ID	Test case	Step	Description	Expected Result	Results
1	Create an experiment in an empty database	1	Clear all data in DBMS-UI.	In the DBMS-UI, the experiment table has a new item with name of "Test 01".	Passed
		2	Create an experiment from the sample config with the name "Test 01" and empty description.		
		3	Check the DBMS-UI.		
2	Run an experiment for one epoch	1	Clear all data in DBMS-UI.	In the DBMS-UI, the experiment table has a new item with name of "Test 01". The session table has a new item, with the *experiment* field having the same value as the *id* of the newly added experiment. The *curr_epoch* field has a value of 1. The perf_log table has a new item, with the *session* field having the same value as the *id of the newly added session.*	Passed
		2	Create an experiment from the sample config with the name "Test 01" and empty description.		
		3	Run that experiment for one epoch. No configuration for saving models or predictions are set.		
		4	Check the DBMS-UI.		
3	Run an experiment for ten epochs	1	Clear all data in DBMS-UI.	In the DBMS-UI, the experiment table has a new item with name of "Test 01". The session table has a new item, with the *experiment* field having the same value as the *id* of the newly added experiment. The *curr_epoch* field has a value of 1. The perf_log table has ten new items, each of which contains the *session* field having the same value as the *id* of the newly added session.	Passed
		2	Create an experiment from the sample config with the name "Test 01" and empty description.		
		3	Run that experiment for ten epochs. No configuration for saving models or predictions are set.		
		4	Check the DBMS-UI.		

RESULT	
Total Test Cases:	7
Passed:	7
Fail:	0

TEST REPORT

ID	Test case	Step	Description	Expected Result	Results
4	Continue an experiment for five more epochs	1	Use the results from test case 03.	In the DBMS-UI, the experiment table has no changes. The session table has no new items. The existing session has the *curr_epoch* of 15 . The perf_log table has five new items, each of which contains the *session* field having the same value as the *id* of the existing session.	Passed
		2	Continue the session from test case 03 for five more epochs. No configuration for saving models or predictions are set.		
		3	Check the DBMS-UI.		
5	Run a new session of an experiment.	1	Use the results from test case 04.	In the DBMS-UI, the experiment table has no changes. The session table has one new item, with the *experiment* field having the same value as the *id* of the existing experiment. This new session has the *curr_epoch* of 10. The perf_log table has ten new items, each of which contains the *session* field having the same value as the *id* of the newly added session. The models and predictions tables both have ten new items, each of which contains the session field having the same value as the id of the newly added session, as well as the location of the saved models / predictions.	Passed
		2	Run the experiment created from test 02 for ten epochs. Set saving models and predictions in every epoch.		
		3	Check the DBMS-UI.		

RESULT	
Total Test Cases:	7
Passed:	7
Fail:	0

TEST REPORT

ID	Test case	Step	Description	Expected Result	Results
6	Run a new session of an experiment without continuosly saving models and predictions	1	Use the results from test case 05.	In the DBMS-UI, the experiment table has no changes. The session table has one new item, with the *experiment* field having the same value as the *id* of the existing experiment. This new session has the *curr_epoch* of 10. The perf_log table has ten new items, each of which contains the *session* field having the same value as the *id* of the newly added session. The models and predictions tables both have five new items, each of which contains the session field having the same value as the id of the newly added session, as well as the location of the saved models / predictions. The newly added models and predictions are saved at the 2nd, 4th, 6th, 8th and 10th epoch.	Passed
		2	Run the experiment created from test 02 for ten epochs. Set saving models and predictions in every 2 epochs.		
		3	Check the DBMS-UI.		
7	Create and run a new experiment in a database with existing data	1	Use the results from test case 06.	In the DBMS-UI, the experiment table has one more new item. The session table has one new item, with the *experiment* field having the same value as the *id* of the newly added experiment. This new session has the *curr_epoch* of 10. The perf_log table has ten new items, each of which contains the *session* field having the same value as the *id* of the newly added session. These ten new items has different metrics field than the existing items.	Passed
		2	Create an experiment from a modified version of the sample config where different metrics were used. This experiment has the name of "Test 02" and the description of "Modified Config".		
			Run that experiment created from for ten epochs. No configuration for saving models and predictions are set.		
		3	Check the DBMS-UI.		

Appendix D

The detailed U-net model structure

Table D.1: Overview of the architecture used to train on the head and neck cancer dataset. The downsampling path is given in the table.

Layer (type)	Output Shape	Params	Connected to
input_1 (InputLayer)	(191, 265, 2)	0	
conv2d (Conv2D)	(191, 265, 64)	1216	input_1
batch_normalization (BatchNormalization)	(191, 265, 64)	256	conv2d
conv2d_1 (Conv2D)	(191, 265, 64)	36928	batch_normalization
batch_normalization_1 (BatchNormalization)	(191, 265, 64)	256	conv2d_1
max_pooling2d (MaxPooling2D)	(95, 132, 64)	0	batch_normalization_1
conv2d_2 (Conv2D)	(95, 132, 128)	73856	max_pooling2d
batch_normalization_2 (BatchNormalization)	(95, 132, 128)	512	conv2d_2
conv2d_3 (Conv2D)	(95, 132, 128)	147584	batch_normalization_2
batch_normalization_3 (BatchNormalization)	(95, 132, 128)	512	conv2d_3
max_pooling2d_1 (MaxPooling2D)	(47, 66, 128)	0	batch_normalization_3
conv2d_4 (Conv2D)	(47, 66, 256)	295168	max_pooling2d_1
batch_normalization_4 (BatchNormalization)	(47, 66, 256)	1024	conv2d_4
conv2d_5 (Conv2D)	(47, 66, 256)	590080	batch_normalization_4
batch_normalization_5 (BatchNormalization)	(47, 66, 256)	1024	conv2d_5
max_pooling2d_2 (MaxPooling2D)	(23, 33, 256)	0	batch_normalization_5
conv2d_6 (Conv2D)	(23, 33, 512)	1180160	max_pooling2d_2
batch_normalization_6 (BatchNormalization)	(23, 33, 512)	2048	conv2d_6
conv2d_7 (Conv2D)	(23, 33, 512)	2359808	batch_normalization_6
batch_normalization_7 (BatchNormalization)	(23, 33, 512)	2048	conv2d_7
max_pooling2d_3 (MaxPooling2D)	(11, 16, 512)	0	batch_normalization_7

Each convolutional layer is followed by a batch normalization layer.
The max pooling layers reduce the sizes of the image tensor by half.

Table D.2: Overview of the architecture used to train on the head and neck cancer dataset. The bottleneck is given in the table.

Layer (type)	Output Shape	Params	Connected to
conv2d_8 (Conv2D)	(11, 16, 1024)	4719616	max_pooling2d_3
batch_normalization_8 (BatchNormalization)	(11, 16, 1024)	4096	conv2d_8
conv2d_9 (Conv2D)	(11, 16, 1024)	9438208	batch_normalization_8
batch_normalization_9 (BatchNormalization)	(11, 16, 1024)	4096	conv2d_9

The bottle neck contains two convolutional layers, each followed by a batch-normalization layer.

Table D.3: Overview of the architecture used to train on the head and neck cancer dataset. The upsampling path is given in the table.

Layer (type)	Output Shape	Params	Connected to
conv2d_transpose (Conv2DTranspose)	(11, 16, 512)	4719104	batch_normalization_9
concatenate (Concatenate)	(23, 33, 1024)	0	batch_normalization_7 conv2d_transpose
conv2d_10 (Conv2D)	(23, 33, 512)	4719104	concatenate
batch_normalization_10 (BatchNormalization)	(23, 33, 512)	2048	conv2d_10
conv2d_11 (Conv2D)	(23, 33, 512)	2359808	batch_normalization_10
batch_normalization_11 (BatchNormalization)	(23, 33, 512)	2048	conv2d_11
conv2d_transpose_1 (Conv2DTranspose)	(23, 33, 256)	1179904	batch_normalization_11
concatenate_1 (Concatenate)	(47, 66, 512)	0	batch_normalization_5 conv2d_transpose_1

The transposed convolutional layers double the sizes of the image tensor.

Table D.4: Overview of the architecture used to train on the head and neck cancer dataset. Continued from Table D.3

Layer (type)	Output Shape	Params	Connected to
conv2d_12 (Conv2D)	(47, 66, 256)	1179904	concatenate_1
batch_normalization_12 (BatchNormalization)	(47, 66, 256)	1024	conv2d_12
conv2d_13 (Conv2D)	(47, 66, 256)	590080	batch_normalization_12
batch_normalization_13 (BatchNormalization)	(47, 66, 256)	1024	conv2d_13
conv2d_transpose_2 (Conv2DTranspose)	(47, 66, 128)	295040	batch_normalization_13
concatenate_2 (Concatenate)	(95, 132, 256)	0	batch_normalization_3 conv2d_transpose_2
conv2d_14 (Conv2D)	(95, 132, 128)	295040	concatenate_2
batch_normalization_14 (BatchNormalization)	(95, 132, 128)	512	conv2d_14
conv2d_15 (Conv2D)	(95, 132, 128)	147584	batch_normalization_14
batch_normalization_15 (BatchNormalization)	(95, 132, 128)	512	conv2d_15
conv2d_transpose_3 (Conv2DTranspose)	(95, 132, 64)	73792	batch_normalization_15
concatenate_3 (Concatenate)	(191, 265, 128)	0	batch_normalization_1 conv2d_transpose_3
conv2d_16 (Conv2D)	(191, 265, 64)	73792	concatenate_3
batch_normalization_16 (BatchNornalization)	(191, 265, 64)	256	conv2d_16
conv2d_17 (Conv2D)	(191, 265, 64)	36928	batch_normalization_16
batch_normalization_17 (BatchNormalization)	(191, 265, 64)	256	conv2d_17
conv2d_18 (Conv2D)	(191, 265, 1)	577	batch_normalization_17

The last layer (conv2d_18) uses the sigmoid activation function to decide if a pixel is part of the cancer tumors.

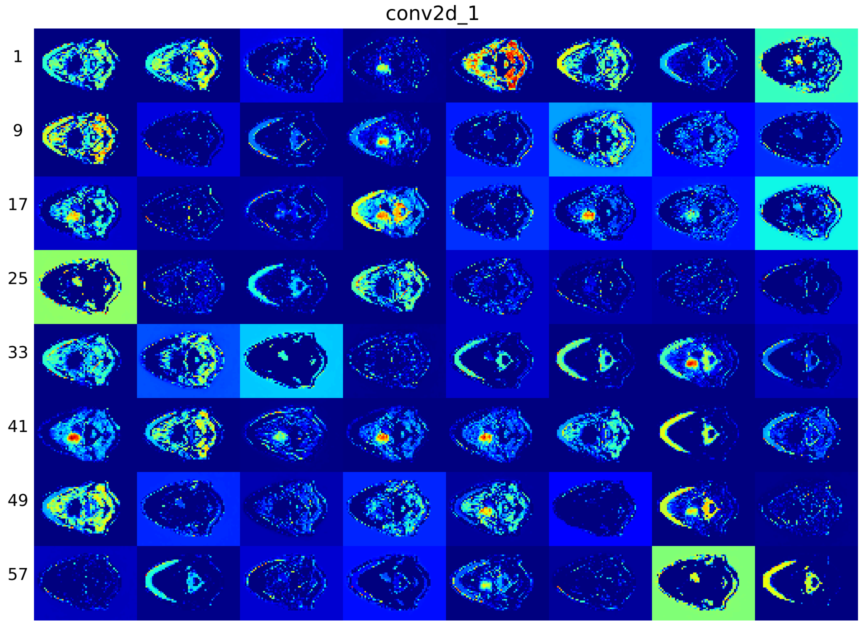
Appendix E

Full activation maps

This part contains the full activation maps of the image slices with high Dice scores as listed in Table 4.2 on page 43.

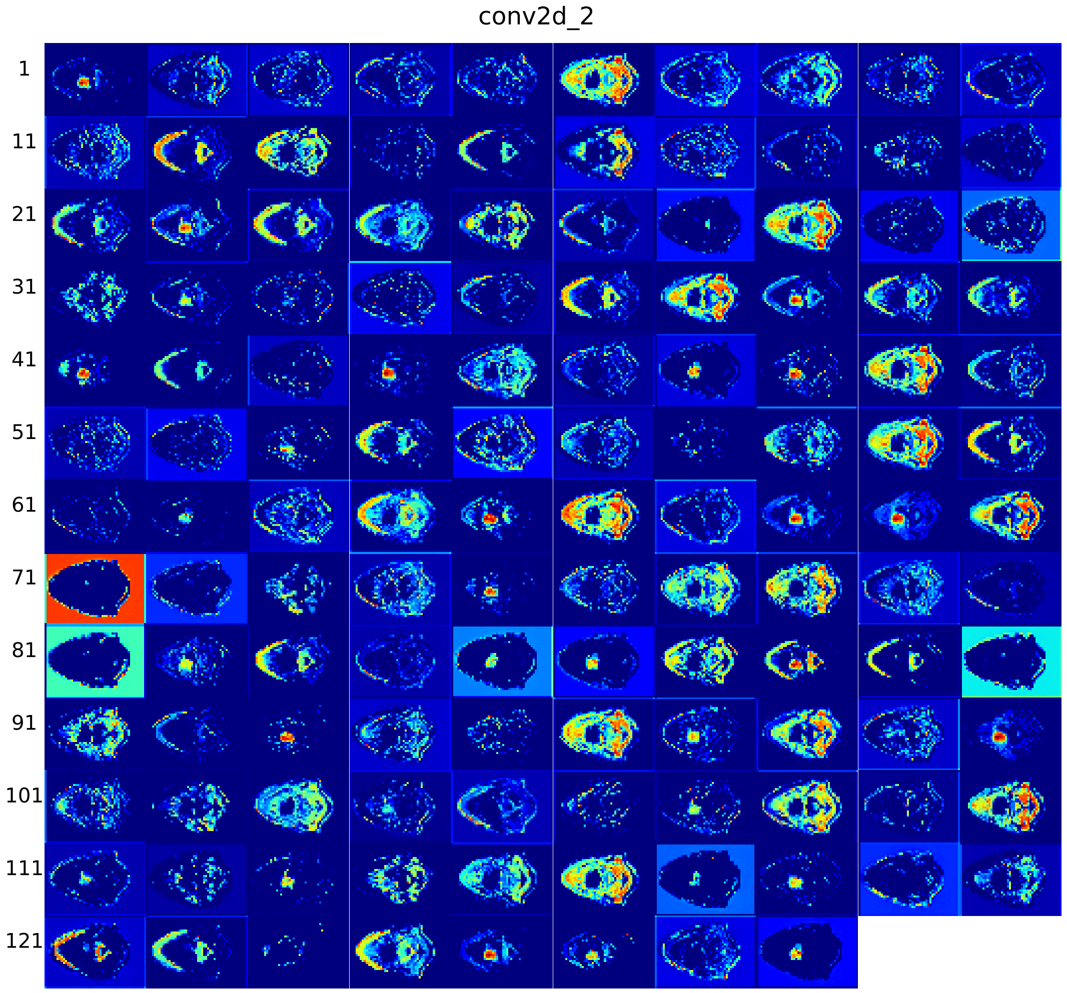
E.1 Activation maps patient 91, slice 86

(a) Layer conv2d



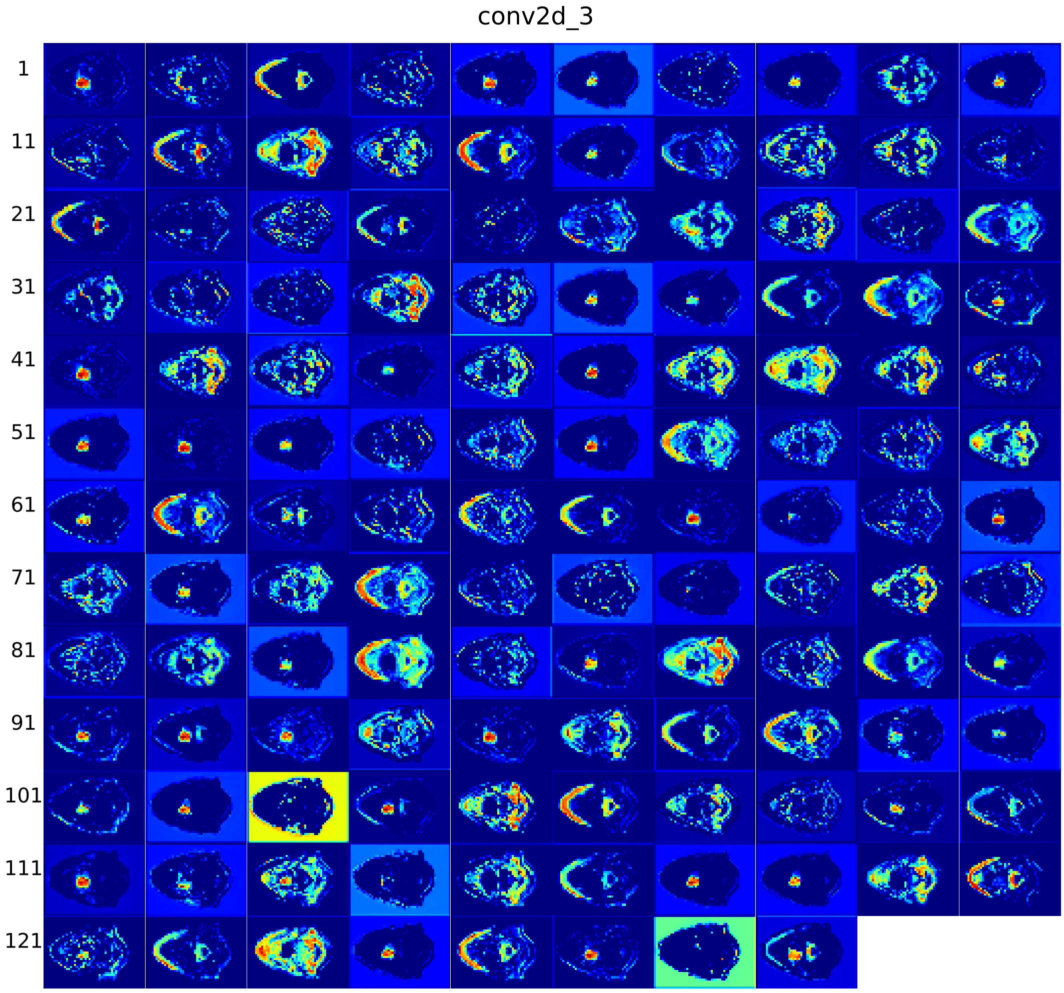
(b) Layer conv2d_1

Figure E.1: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



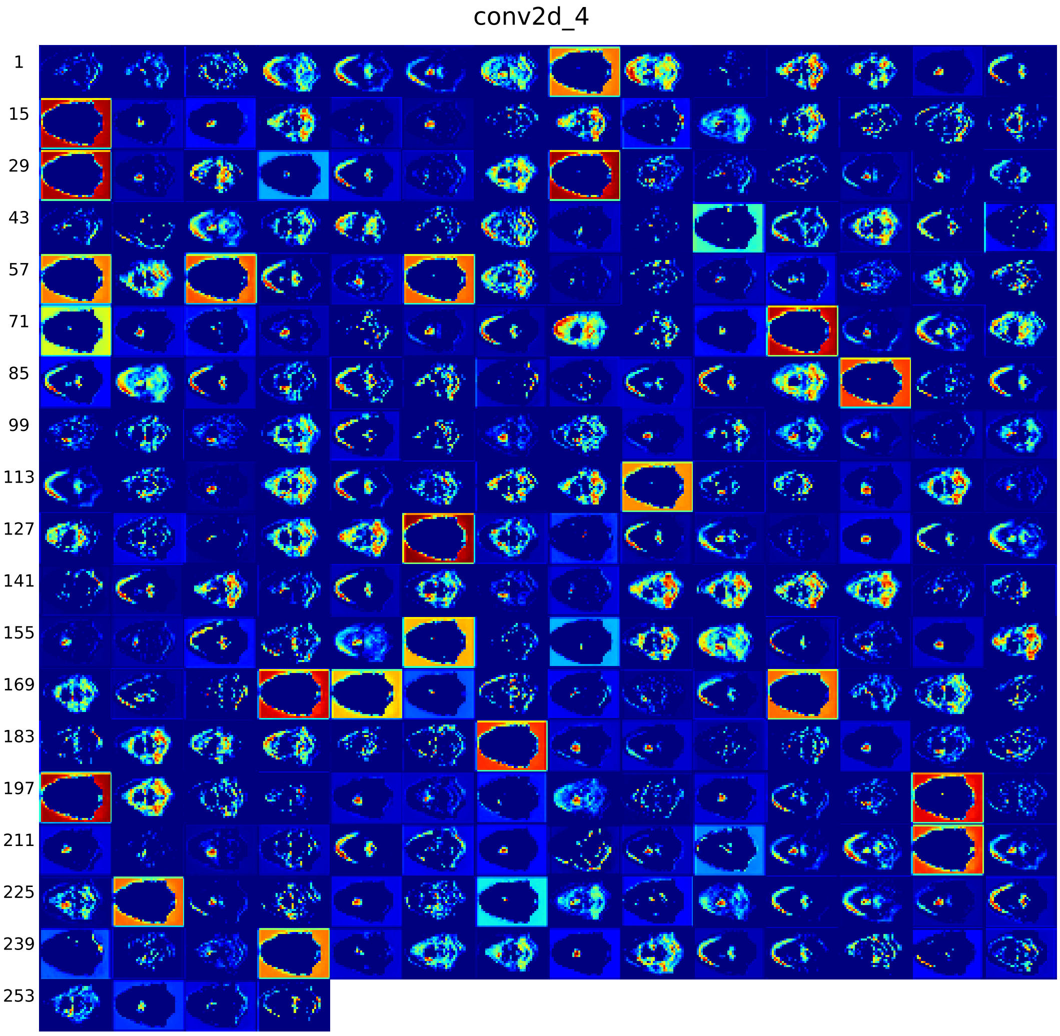
(c) Layer conv2d_2

Figure E.1: Activation maps at conv2d_2 layer (c) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



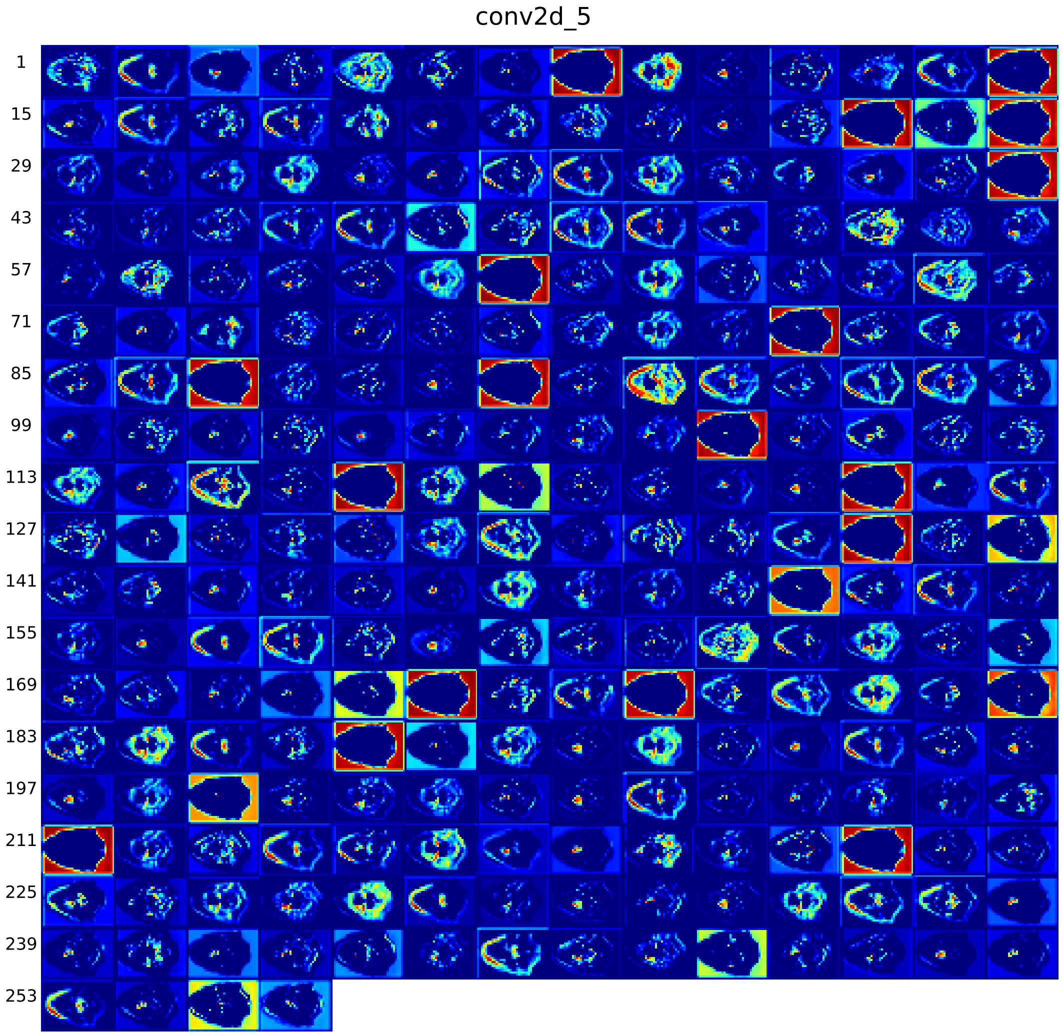
(d) Layer conv2d_3

Figure E.1: Activation maps at conv2d_3 layer (d) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



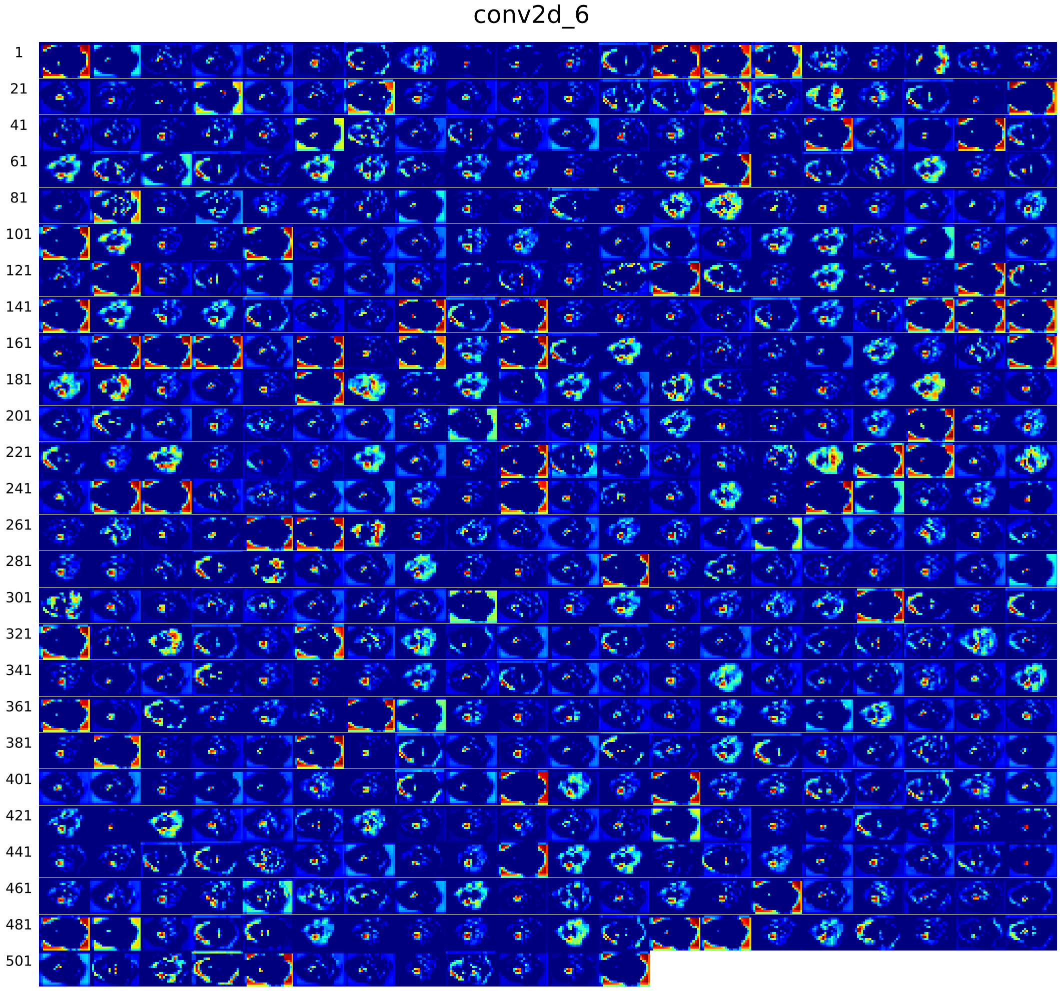
(e) Layer conv2d_4

Figure E.1: Activation maps at conv2d_4 layer (e) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



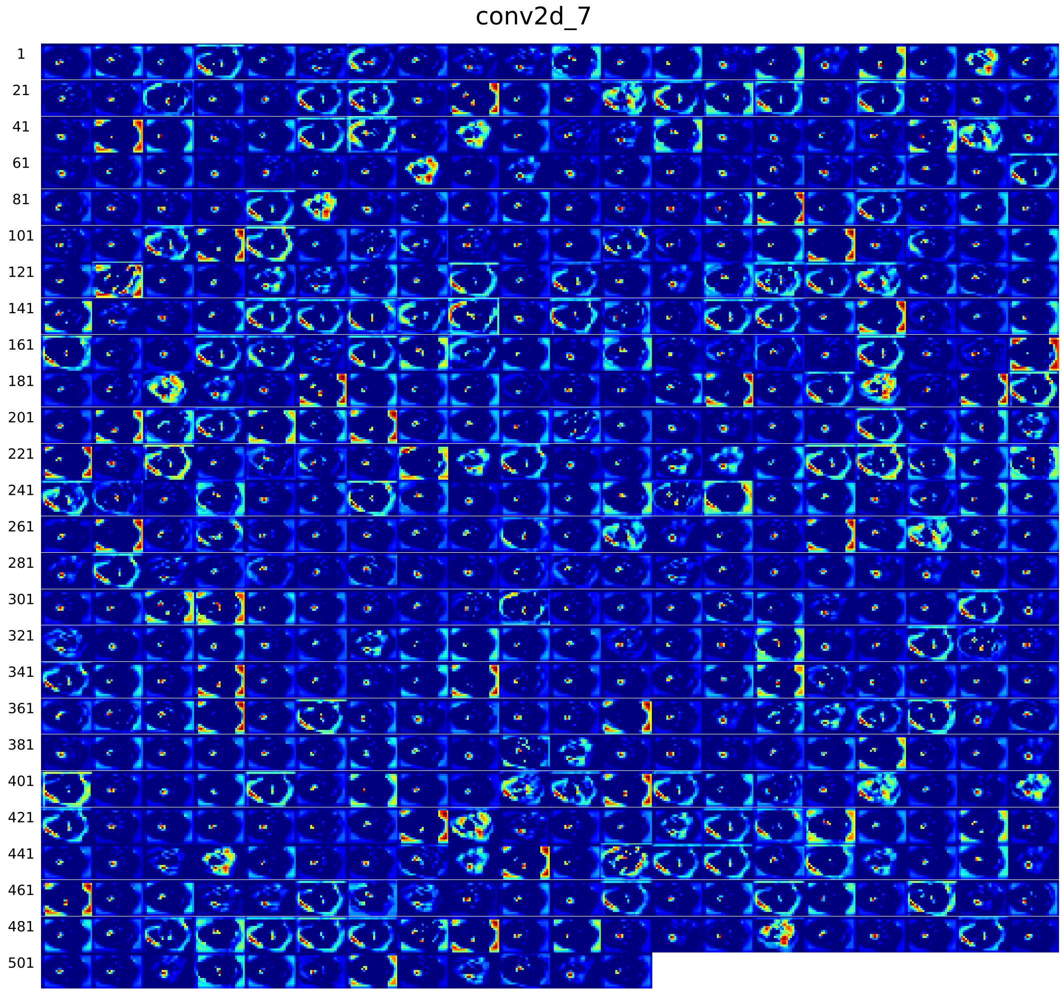
(f) Layer conv2d_5

Figure E.1: Activation maps at conv2d_5 layer (f) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



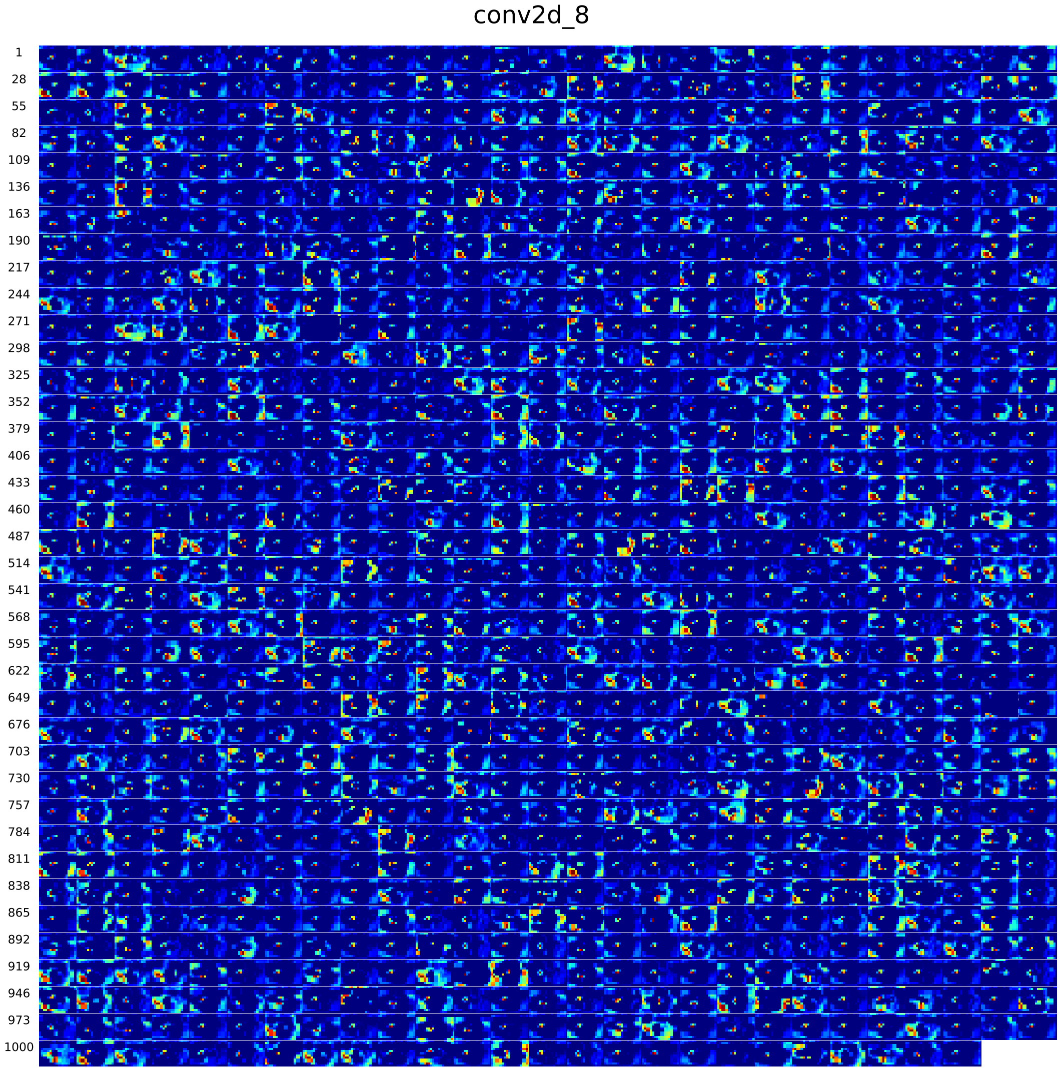
(g) Layer conv2d_6

Figure E.1: Activation maps at conv2d_6 layer (g) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



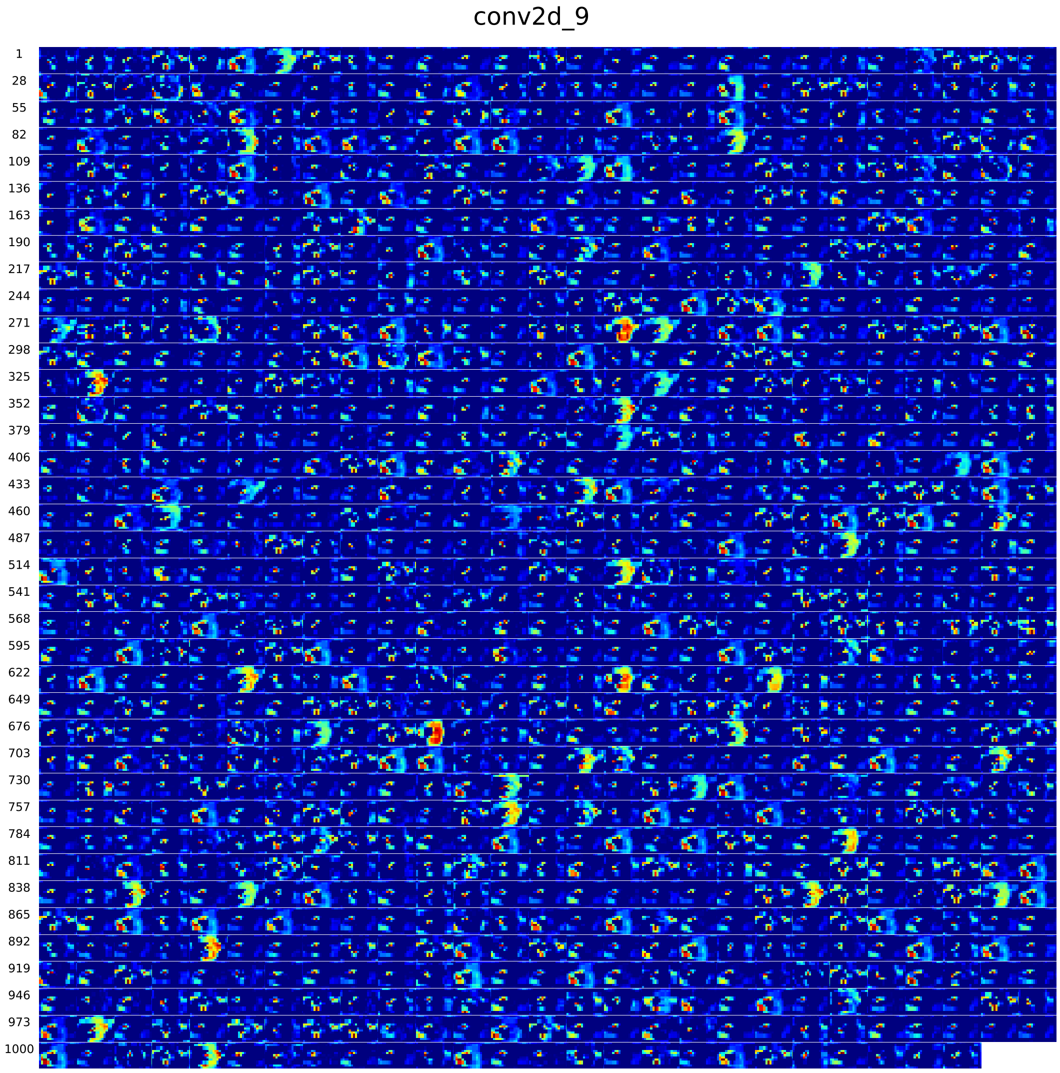
(h) Layer conv2d_7

Figure E.1: Activation maps at conv2d_7 layer (h) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



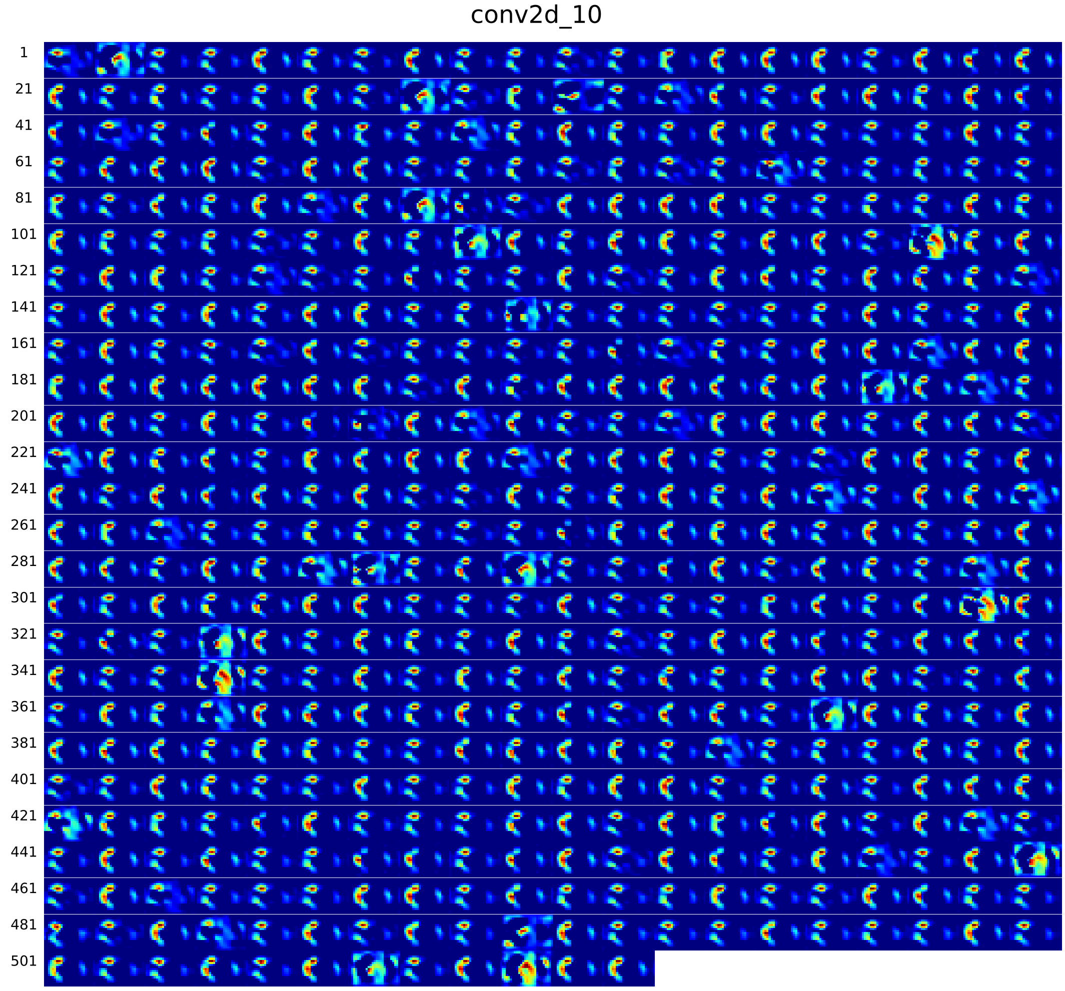
(i) Layer conv2d_8

Figure E.1: Activation maps at conv2d_8 layer (i) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



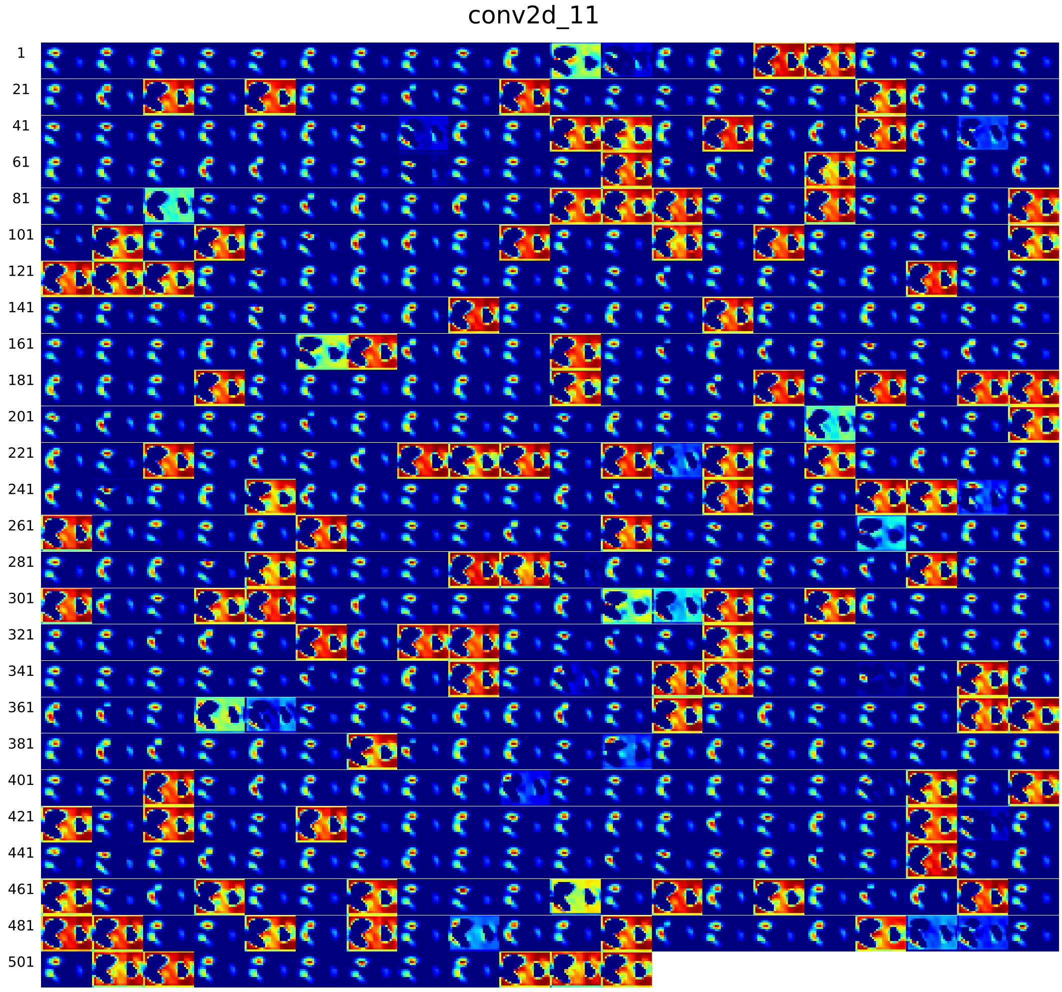
(j) Layer conv2d_9

Figure E.1: Activation maps at conv2d_9 layer (j) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



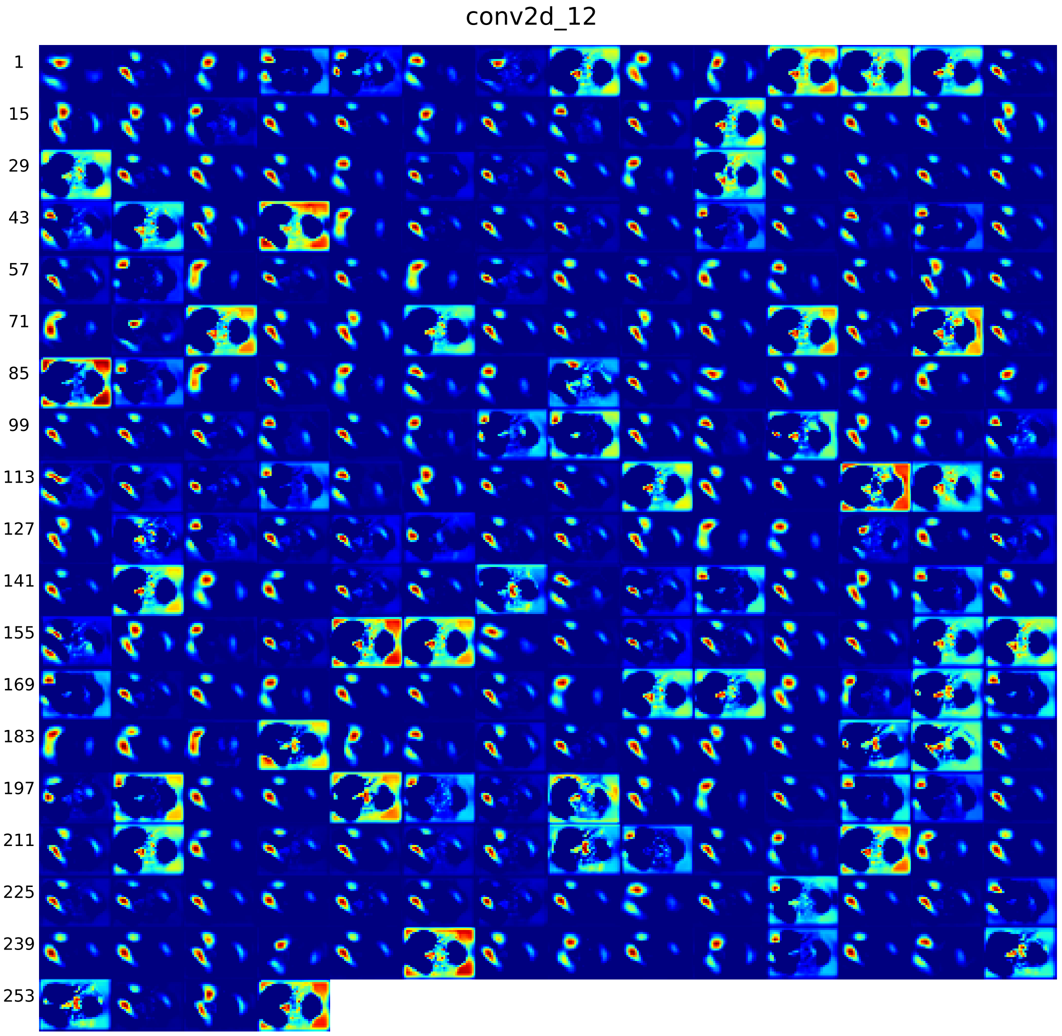
(k) Layer conv2d_10

Figure E.1: Activation maps at conv2d_10 layer (k) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



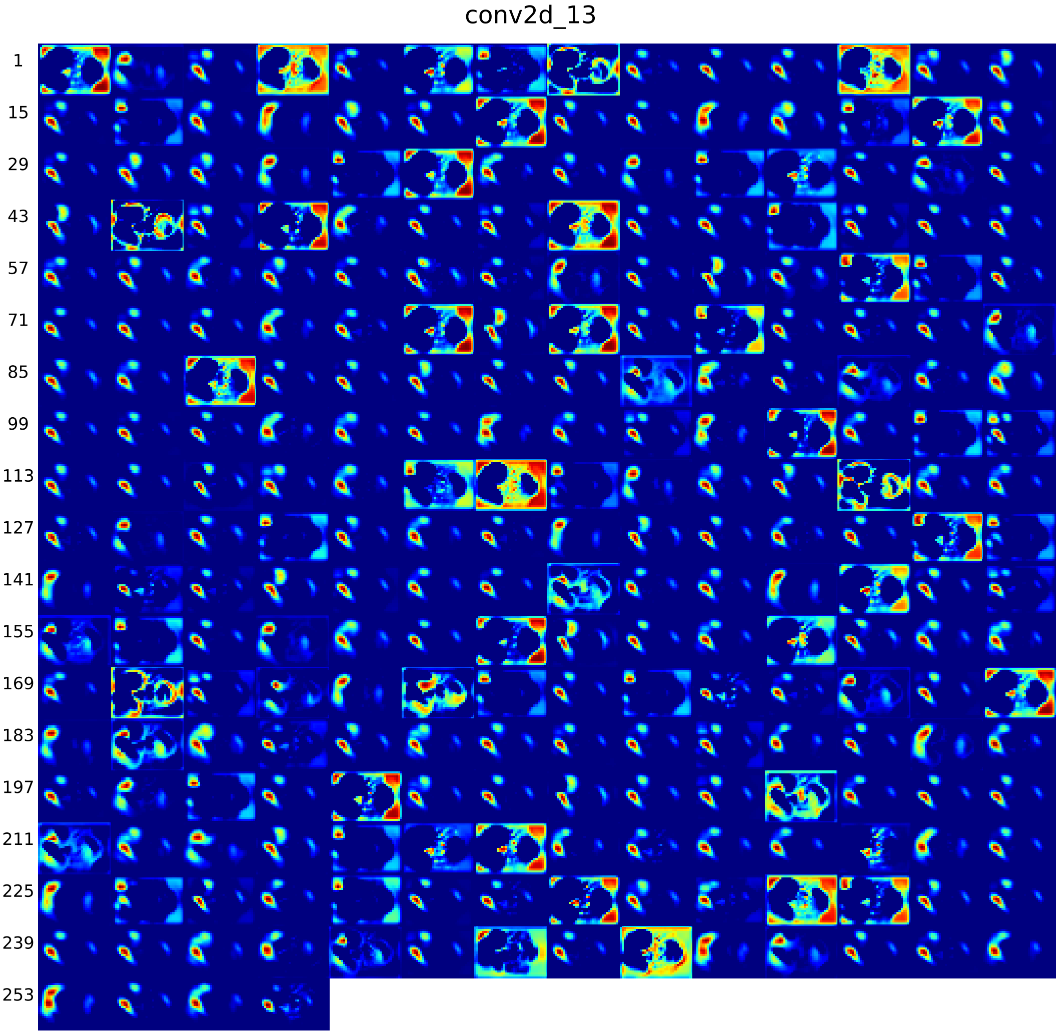
(l) Layer conv2d_11

Figure E.1: Activation maps at conv2d_11 layer (l) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page..



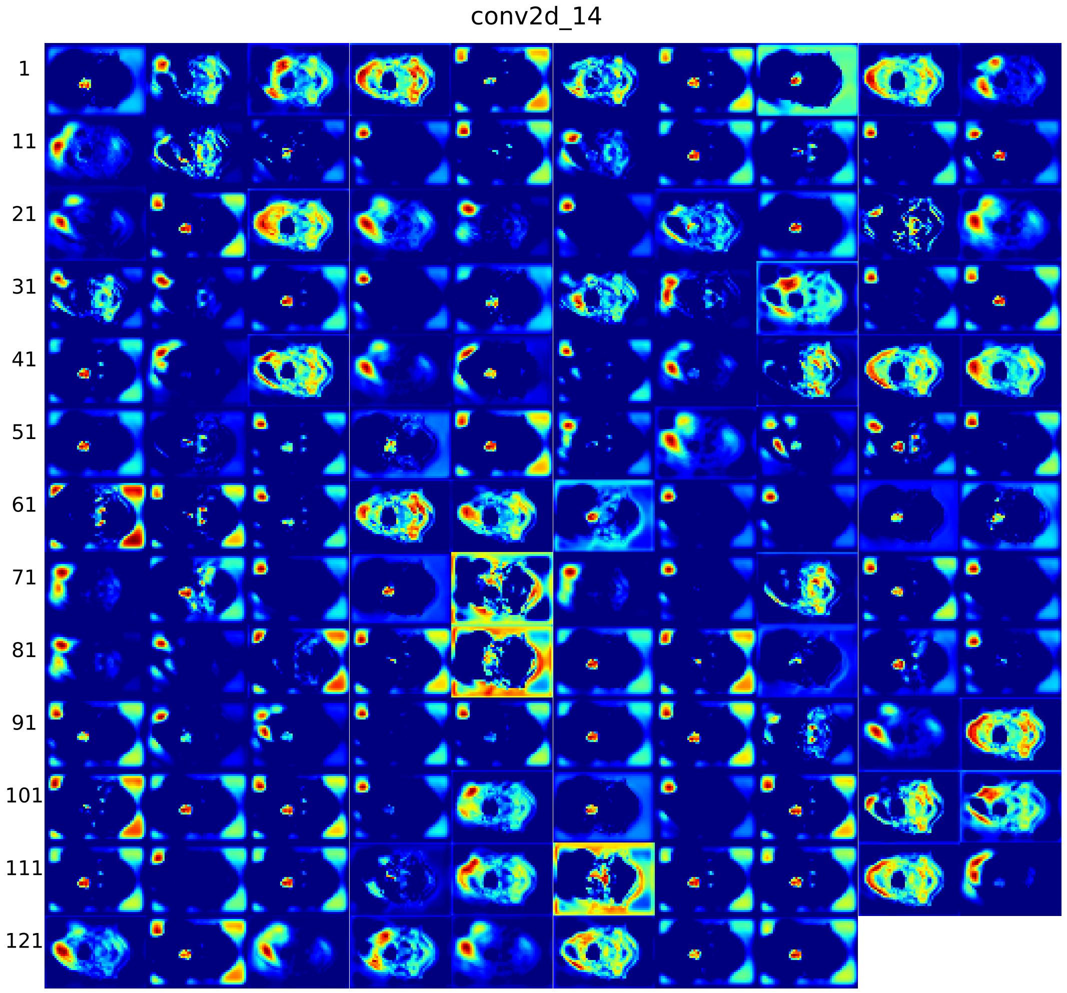
(m) Layer conv2d_12

Figure E.1: Activation maps at conv2d_12 layer (m) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



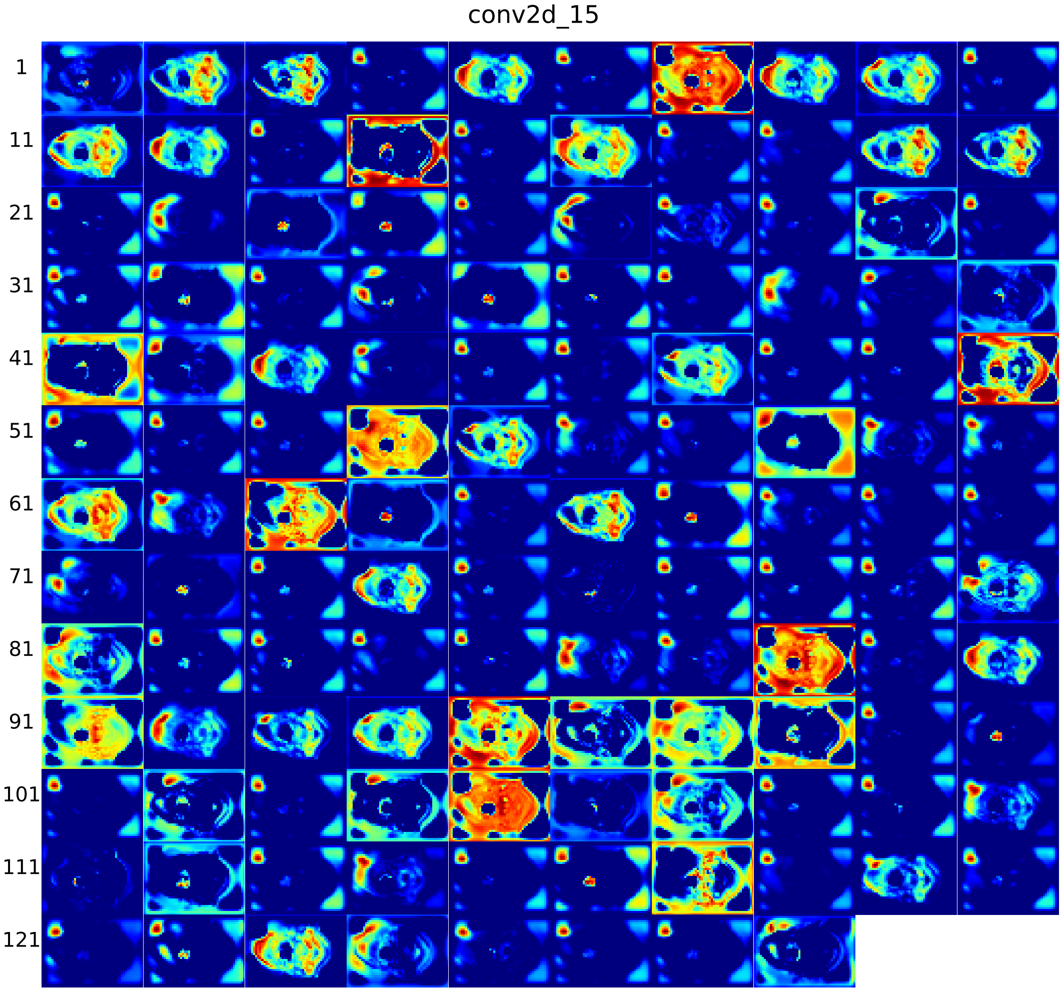
(n) Layer conv2d_13

Figure E.1: Activation maps at conv2d_13 layer (n) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



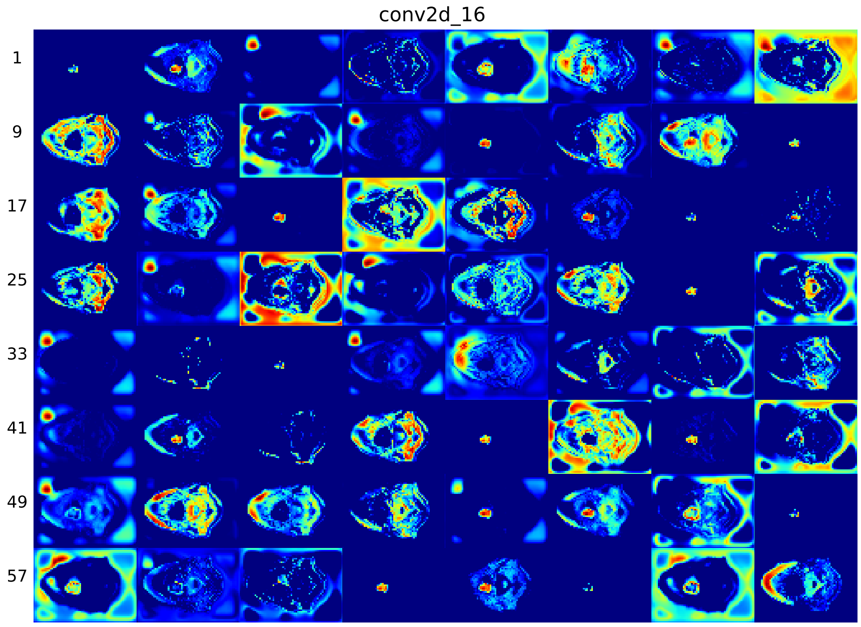
(o) Layer conv2d_14

Figure E.1: Activation maps at conv2d_14 layer (o) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.

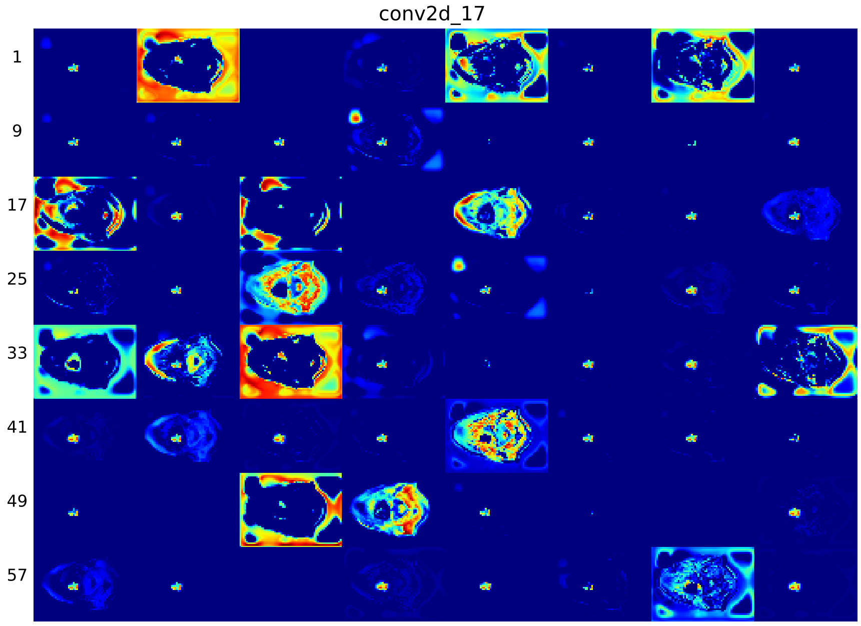


(p) Layer conv2d_15

Figure E.1: Activation maps at conv2d_15 layer (p) of image from patient 91, slice 86, with a Dice score of 0.94. Continued on next page.



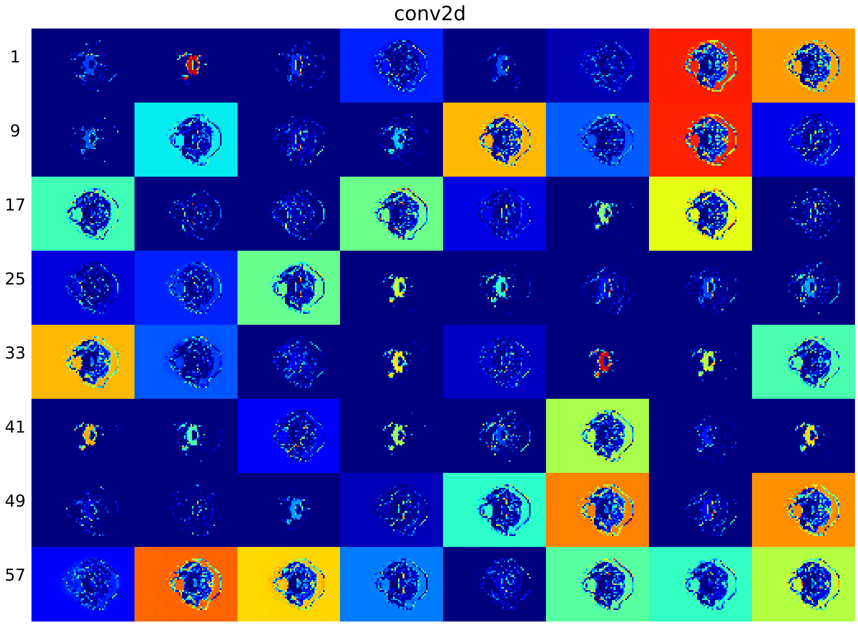
(q) Layer conv2d_16



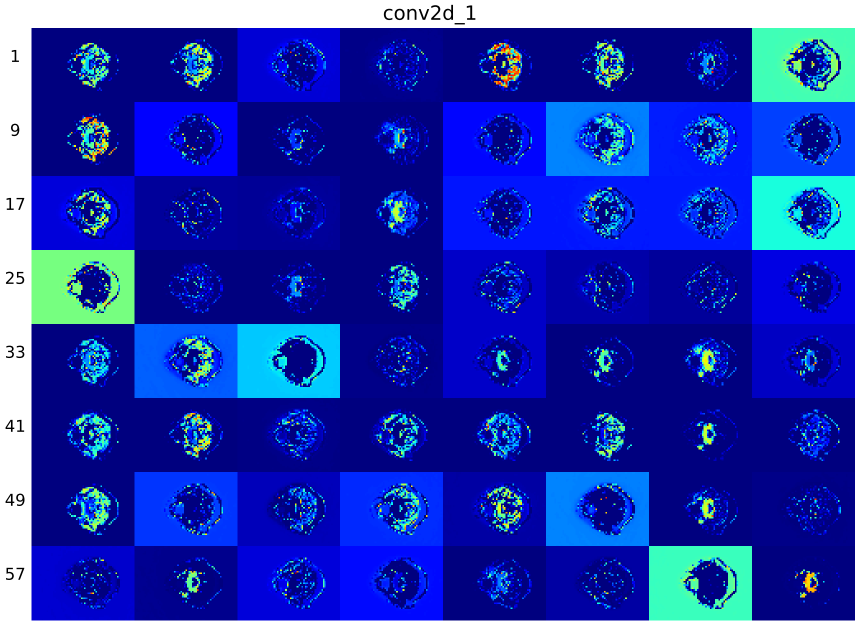
(r) Layer conv2d_17

Figure E.1: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 91, slice 86, with a Dice score of 0.94.

E.2 Activation maps patient 148, slice 11

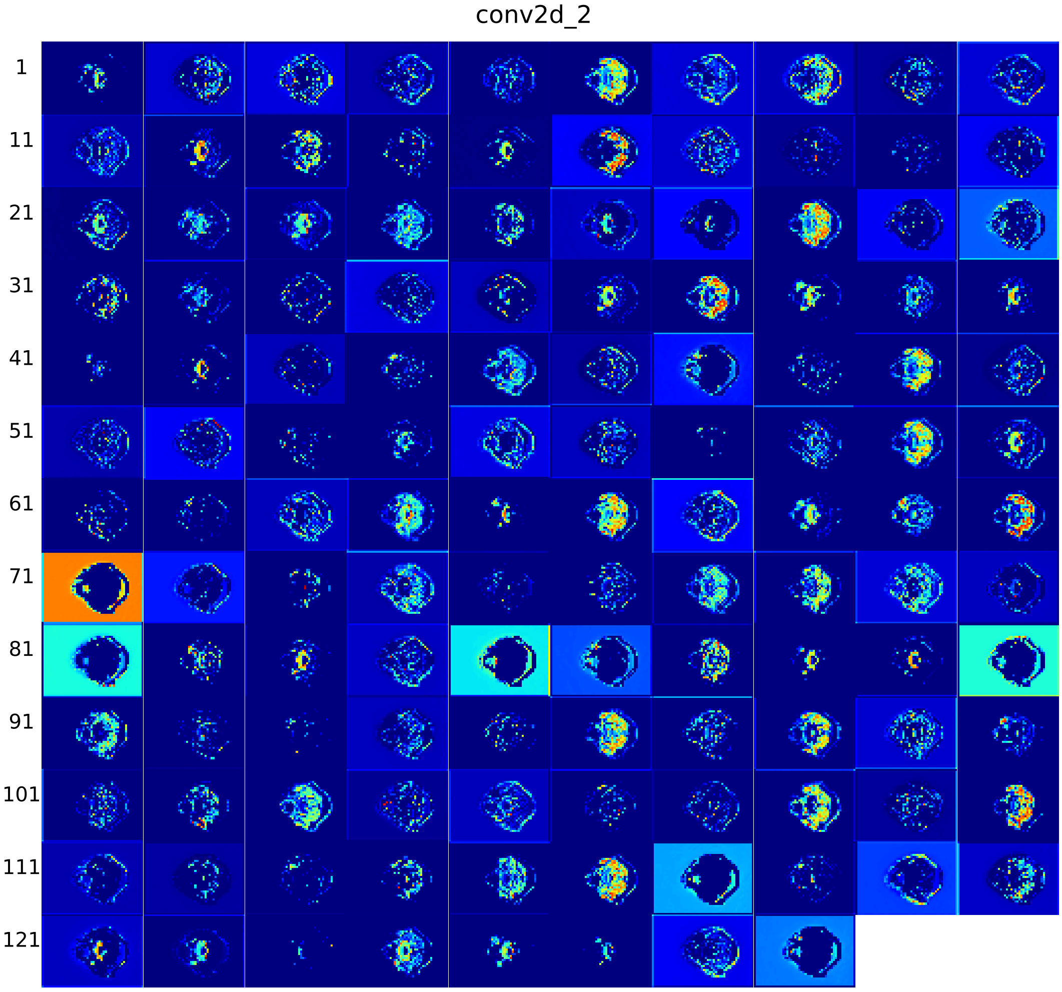


(a) Layer conv2d



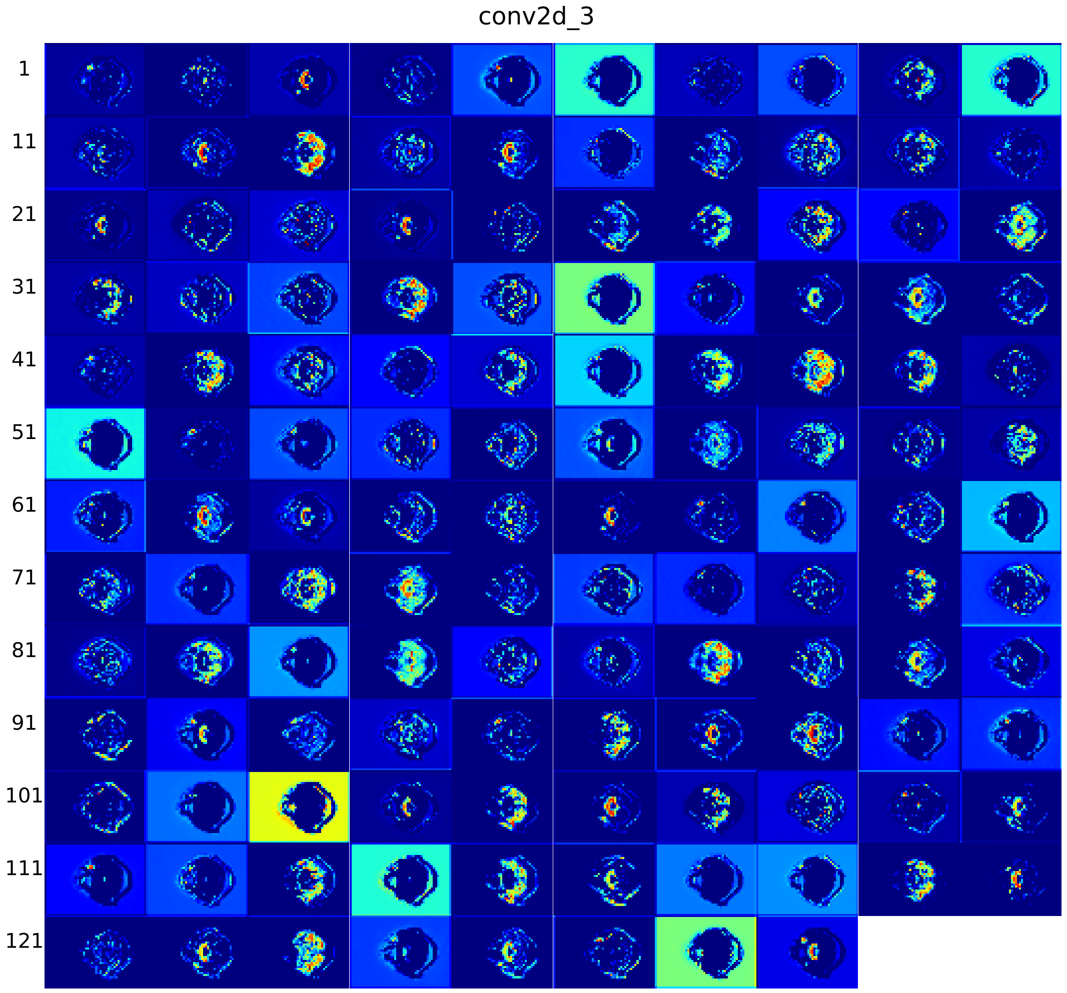
(b) Layer conv2d_1

Figure E.2: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



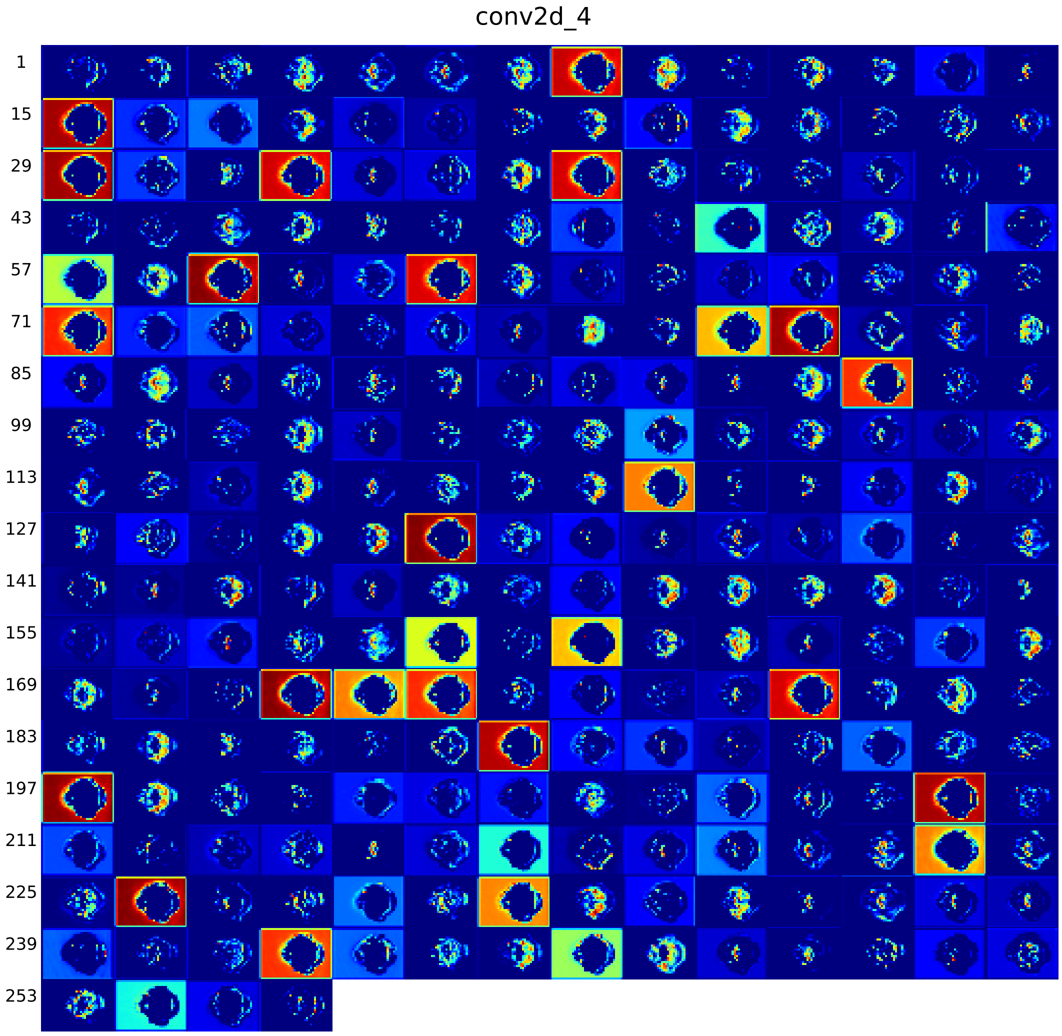
(c) Layer conv2d_2

Figure E.2: Activation maps at conv2d_2 layer (c) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



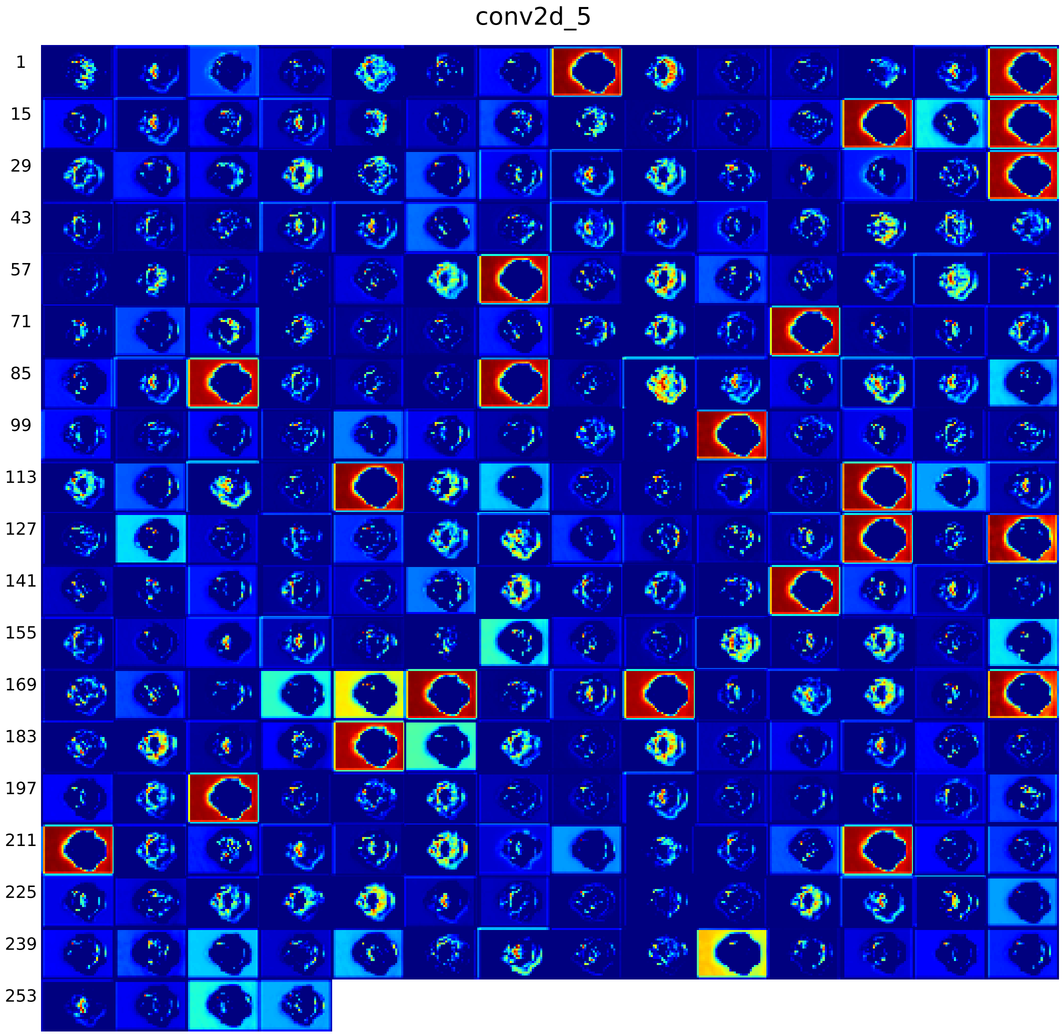
(d) Layer conv2d_3

Figure E.2: Activation maps at conv2d_3 layer (d) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



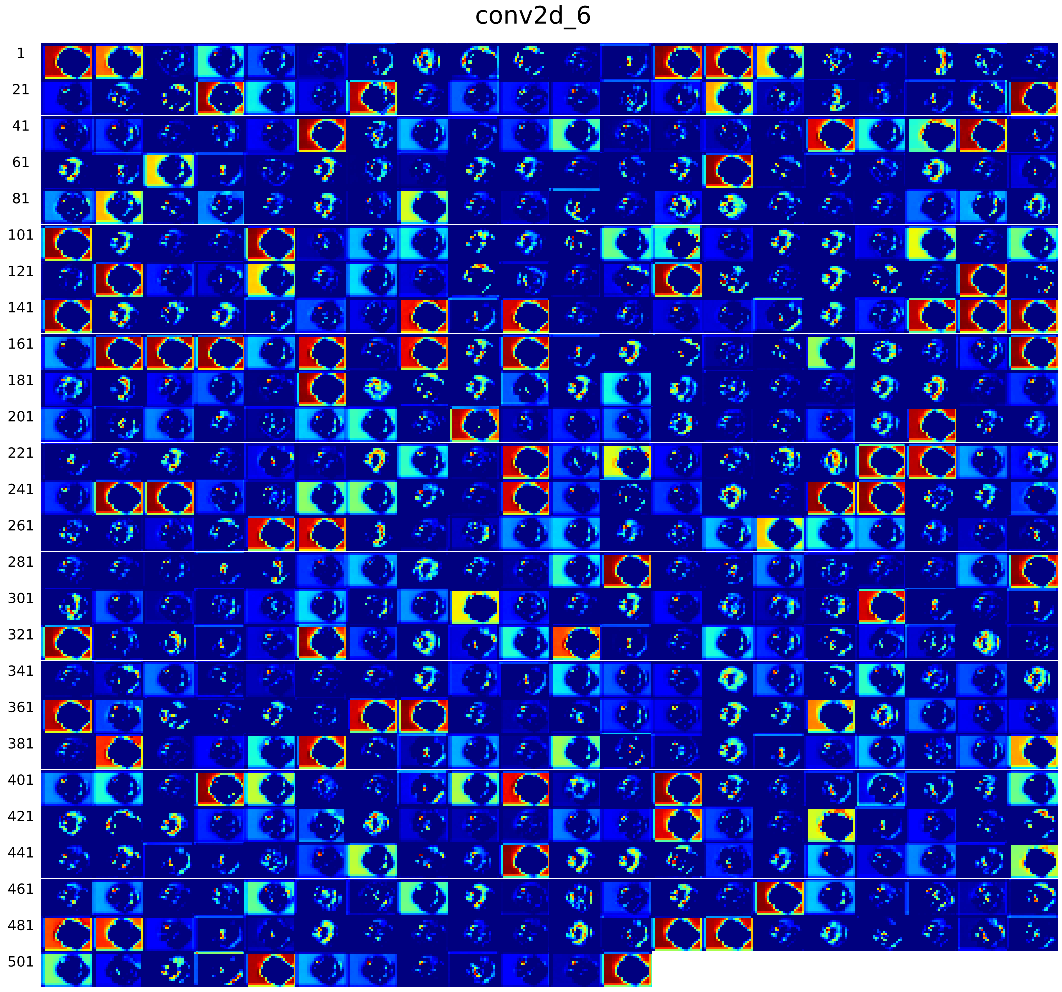
(e) Layer conv2d_4

Figure E.2: Activation maps at conv2d_4 layer (e) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



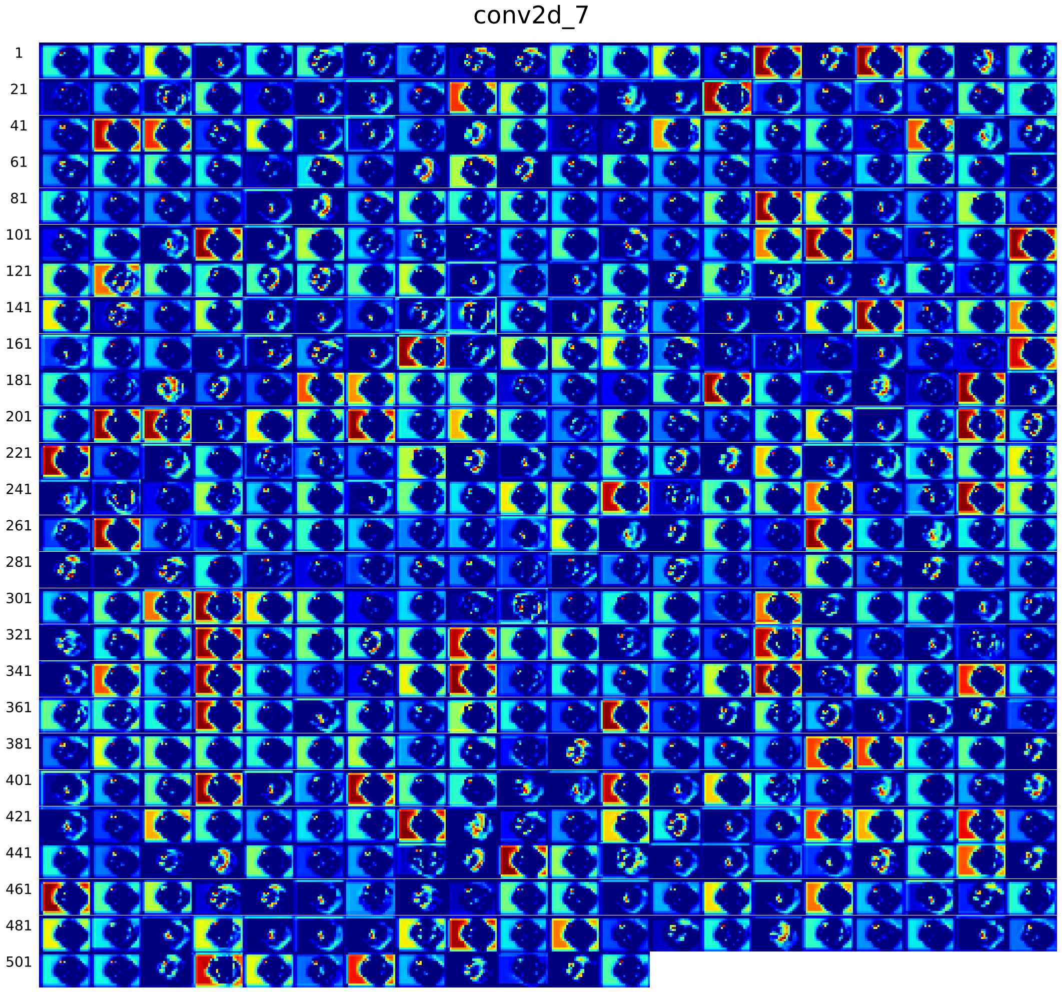
(f) Layer conv2d_5

Figure E.2: Activation maps at conv2d_5 layer (f) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



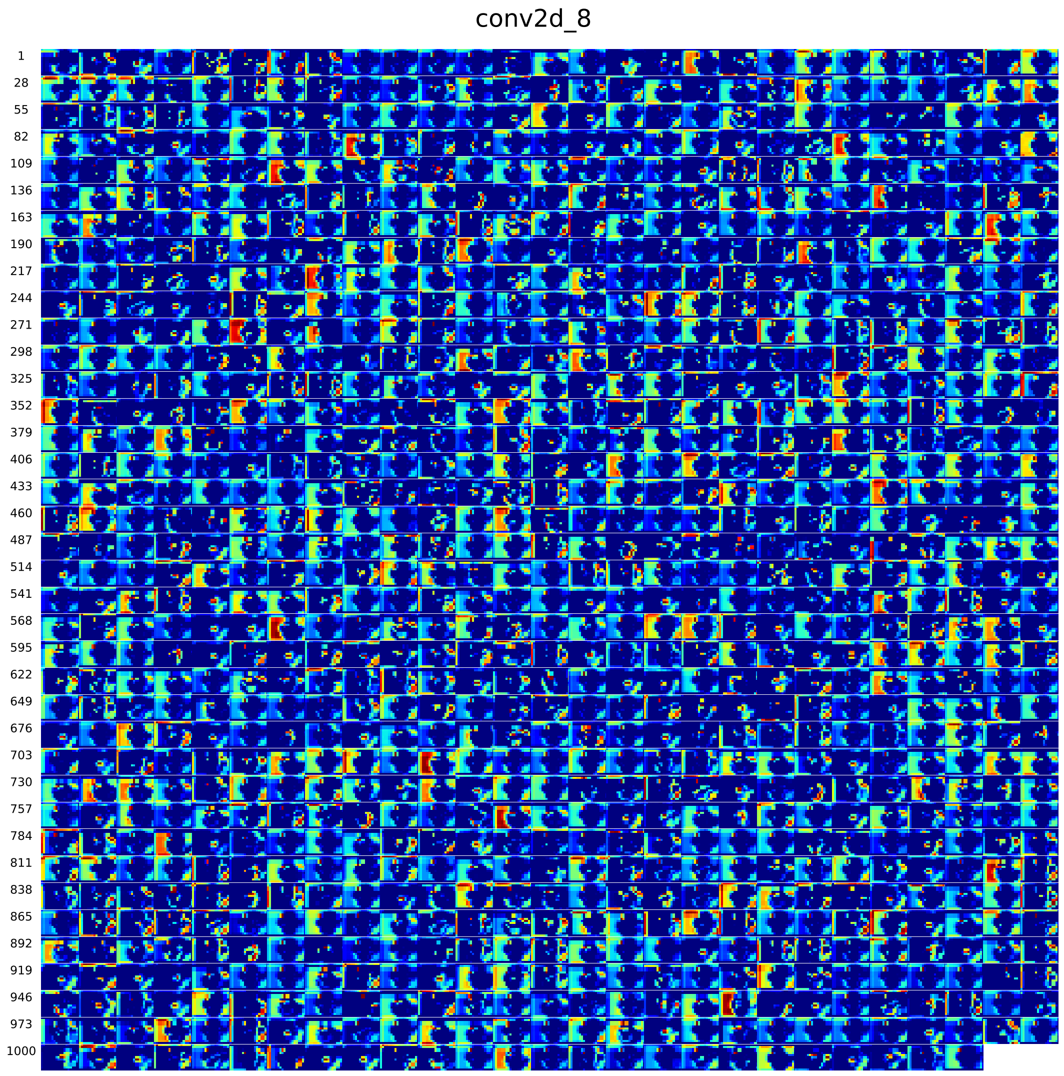
(g) Layer conv2d_6

Figure E.2: Activation maps at conv2d_6 layer (g) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



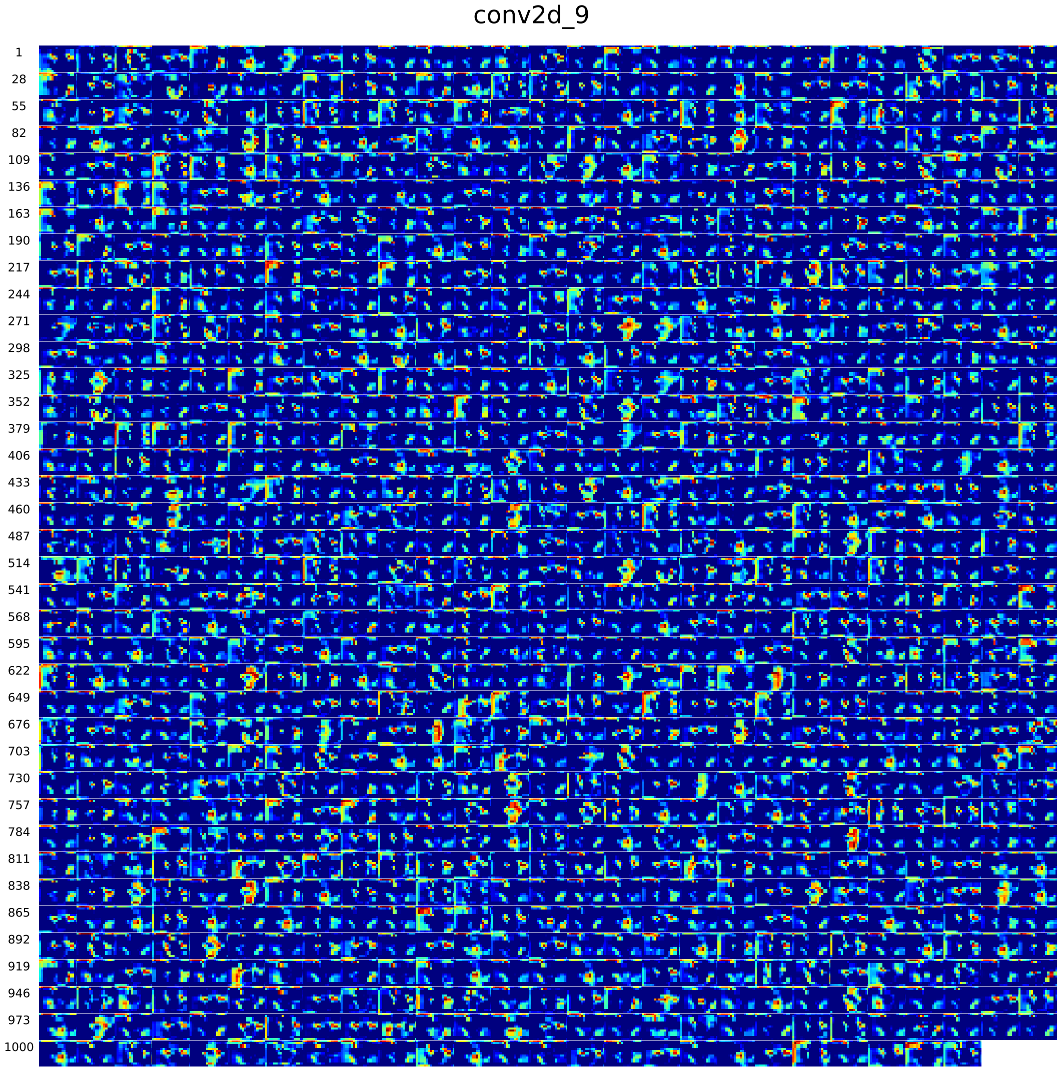
(h) Layer conv2d_7

Figure E.2: Activation maps at conv2d_7 layer (h) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



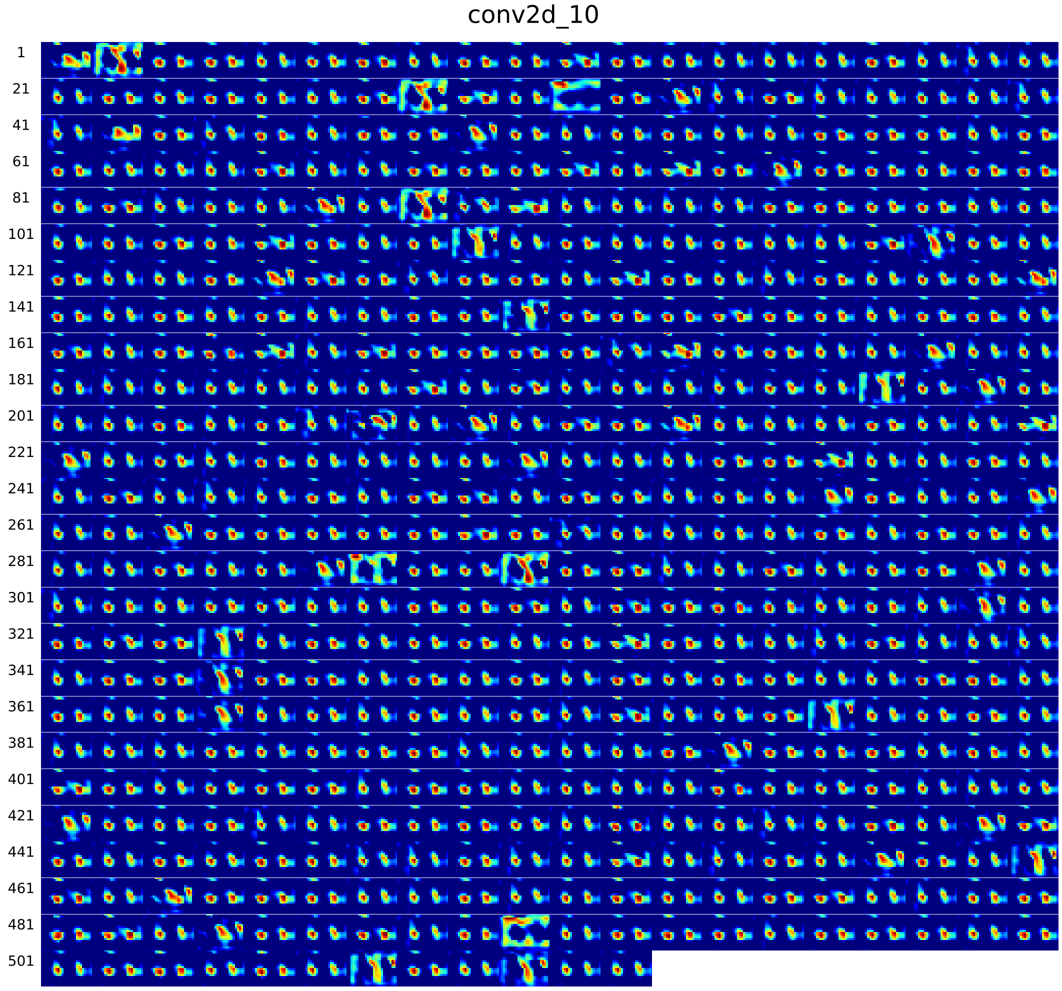
(i) Layer conv2d_8

Figure E.2: Activation maps at conv2d_8 layer (i) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



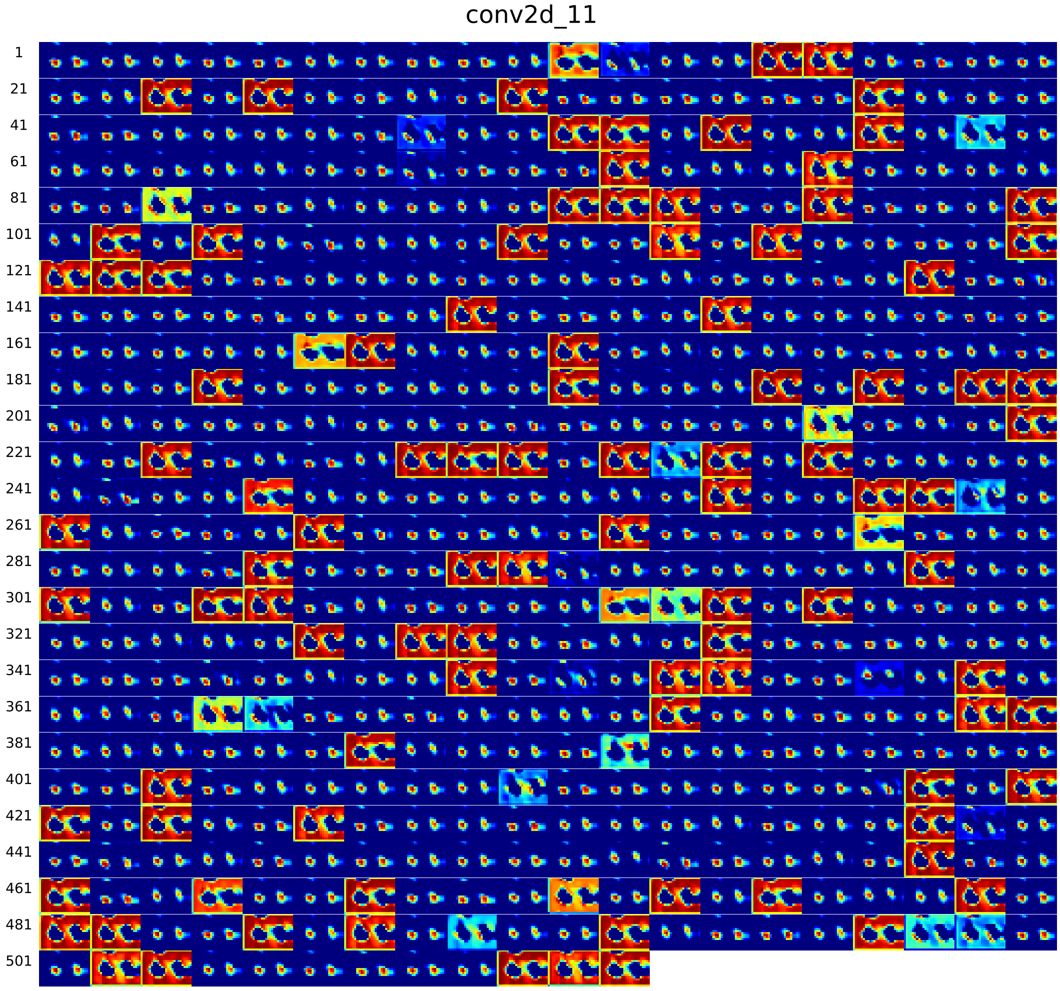
(j) Layer conv2d_9

Figure E.2: Activation maps at conv2d_9 layer (j) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



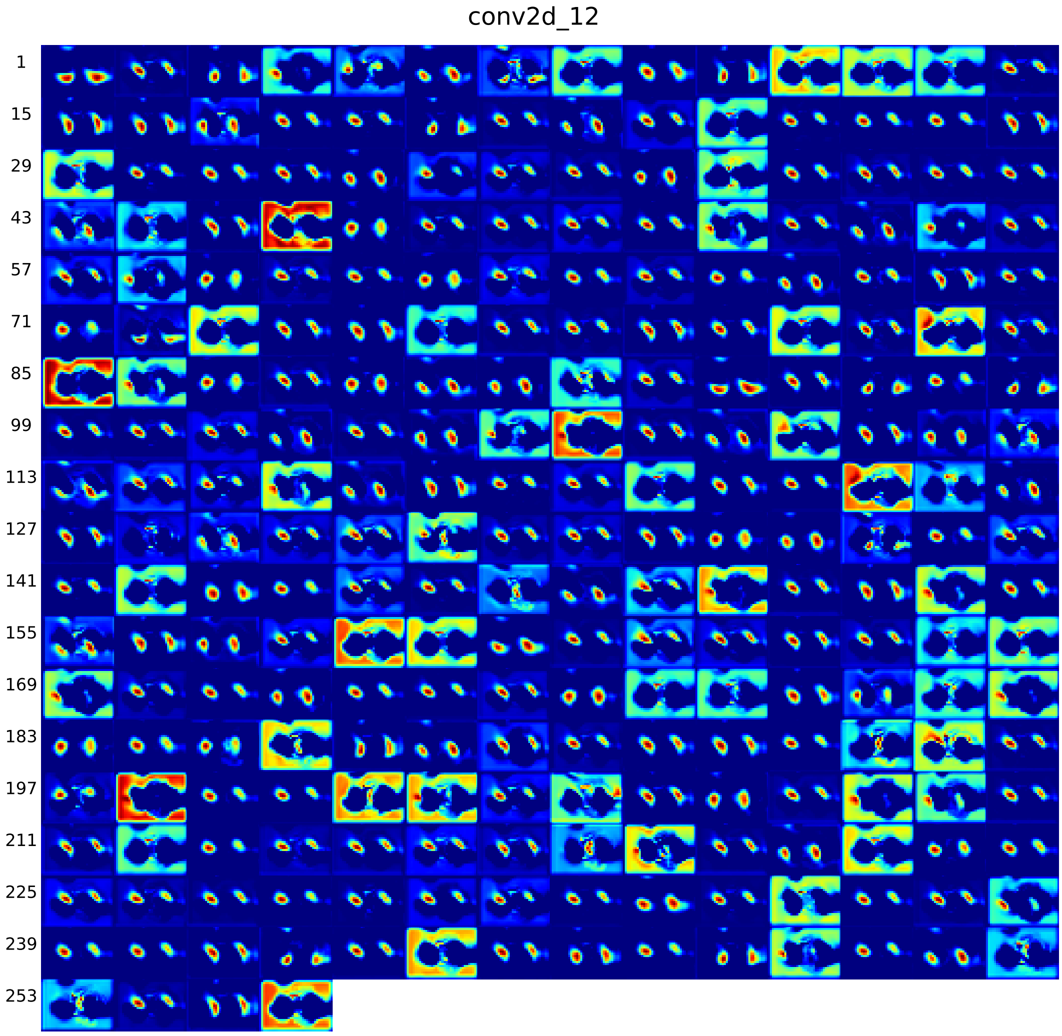
(k) Layer conv2d_10

Figure E.2: Activation maps at conv2d_10 layer (k) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



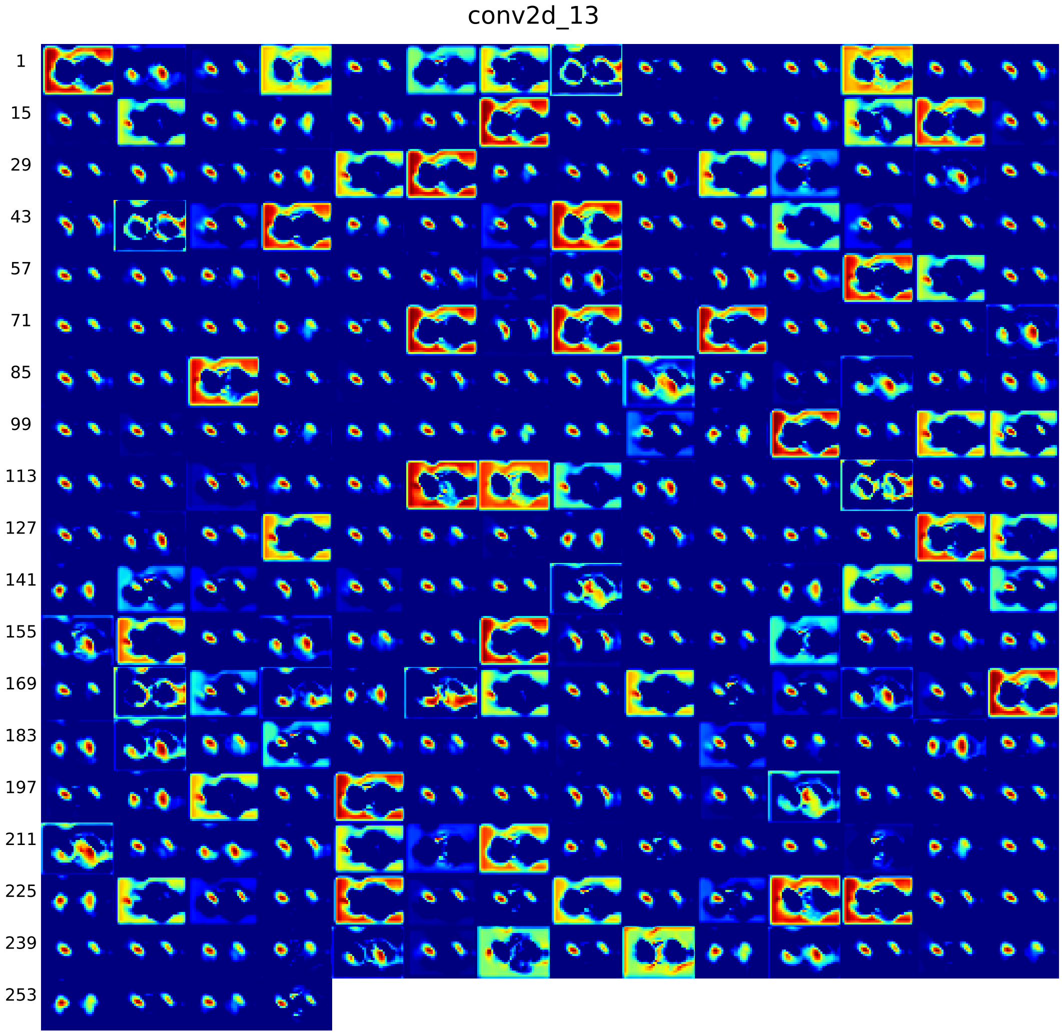
(l) Layer conv2d_11

Figure E.2: Activation maps at conv2d_11 layer (l) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page..



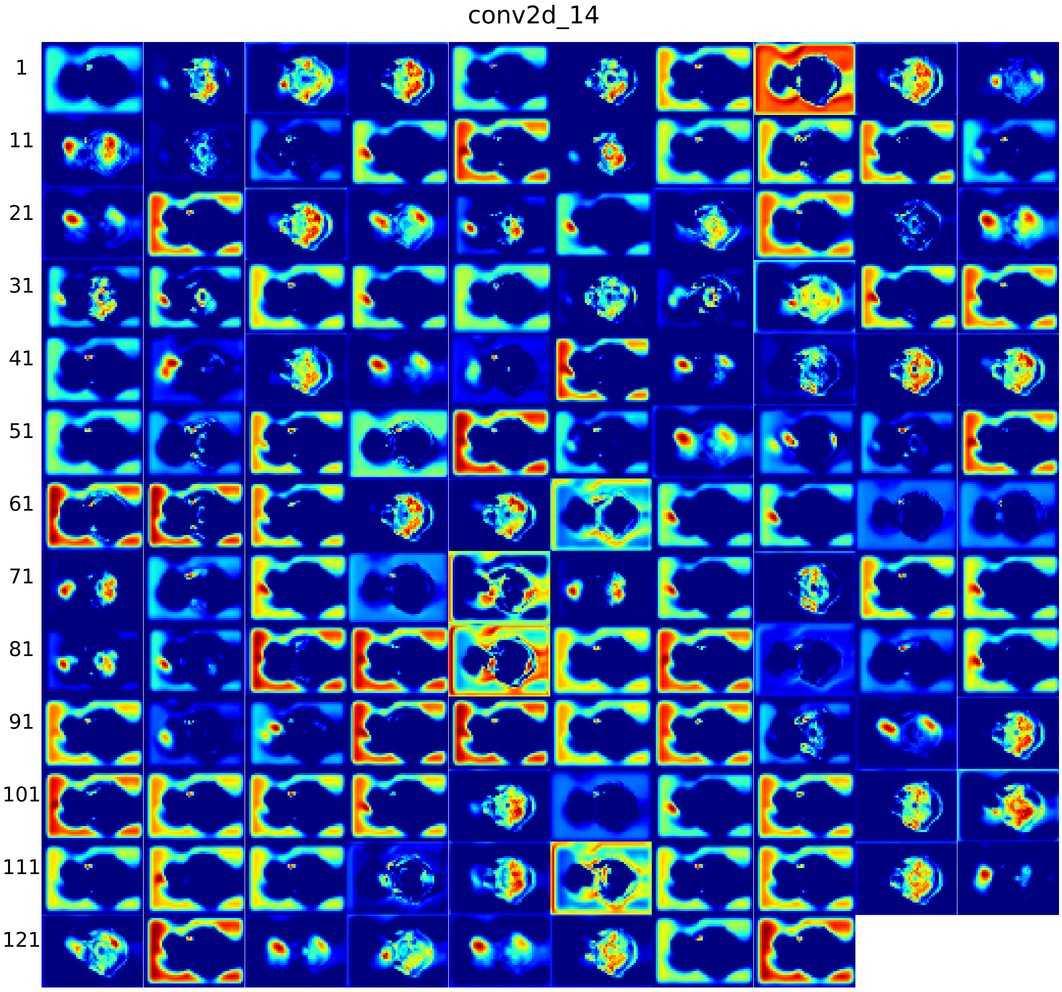
(m) Layer conv2d_12

Figure E.2: Activation maps at conv2d_12 layer (m) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



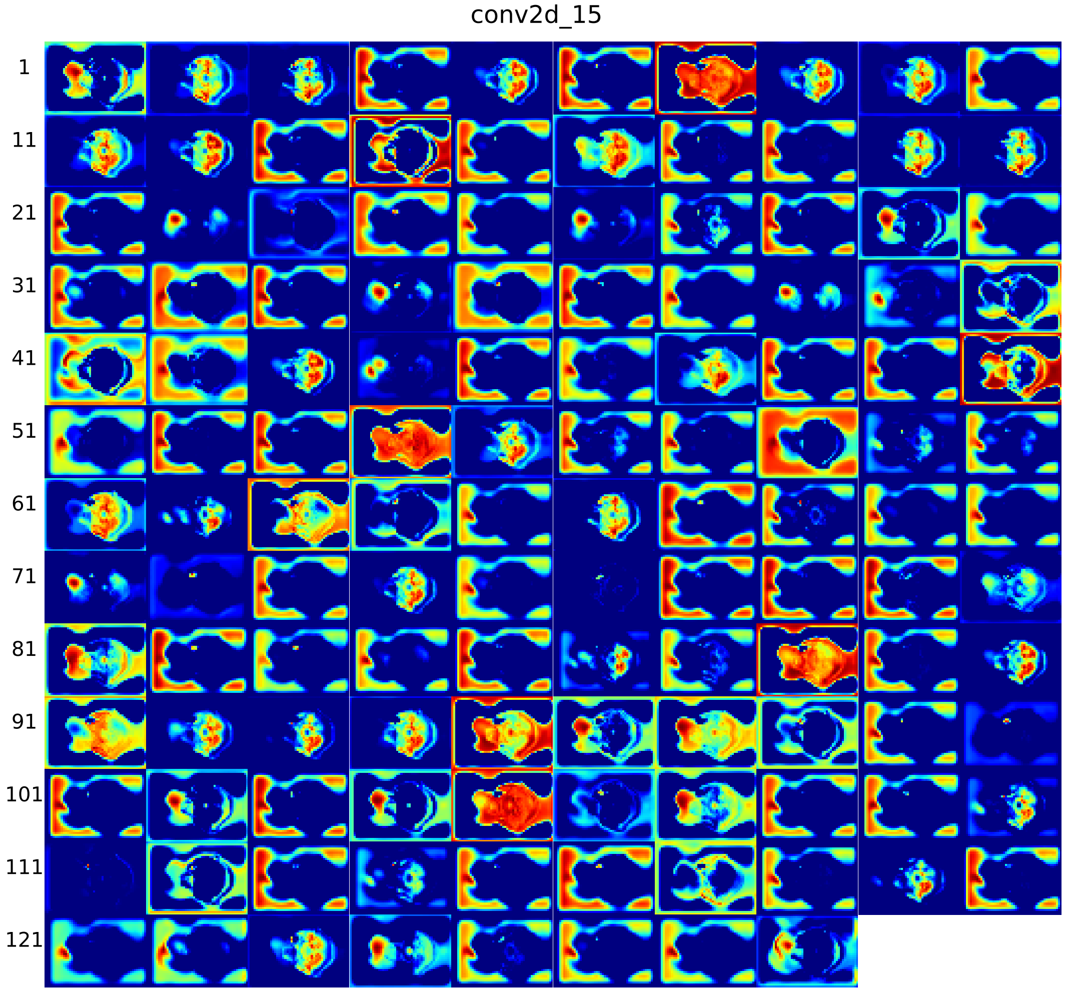
(n) Layer conv2d_13

Figure E.2: Activation maps at conv2d_13 layer (n) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



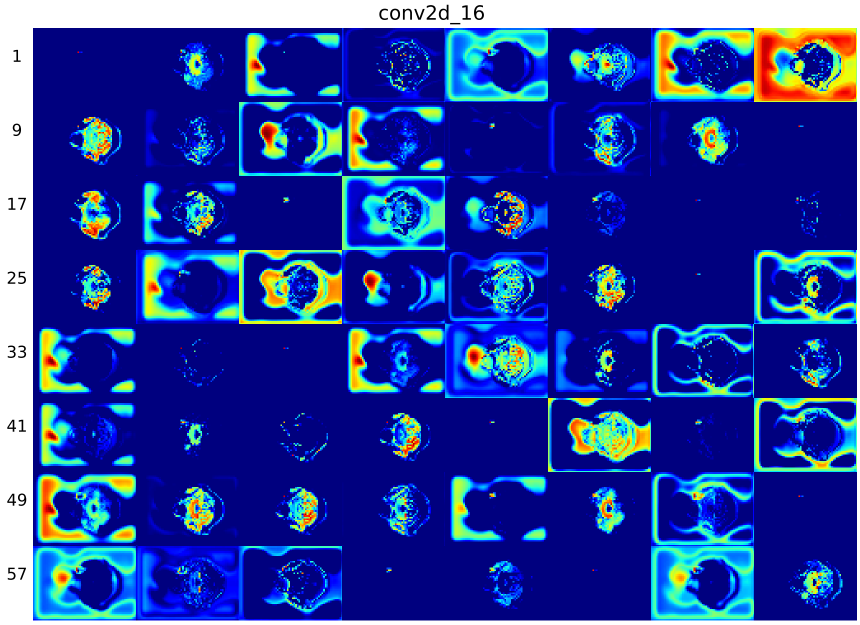
(o) Layer conv2d_14

Figure E.2: Activation maps at conv2d_14 layer (o) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.

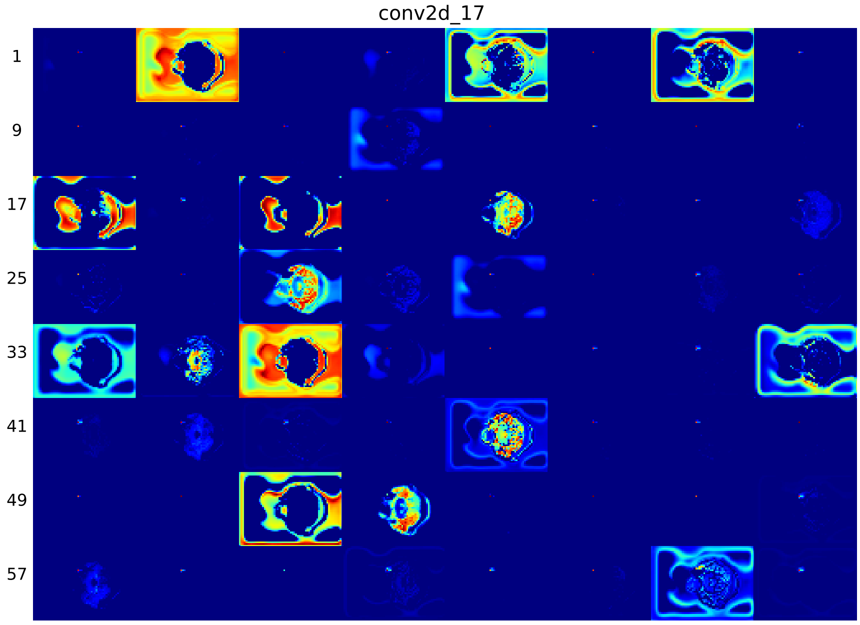


(p) Layer conv2d_15

Figure E.2: Activation maps at conv2d_15 layer (p) of image from patient 148, slice 11, with a Dice score of 0.94. Continued on next page.



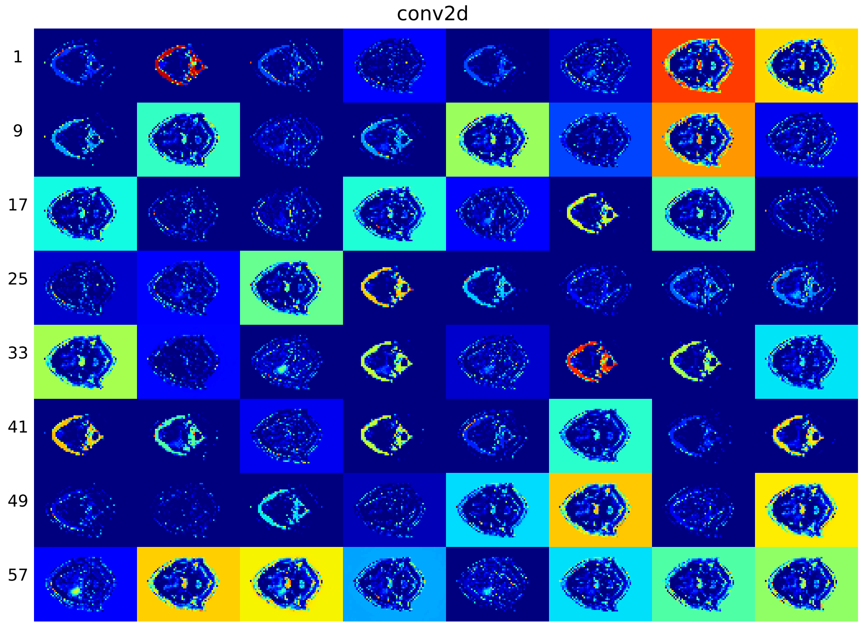
(q) Layer conv2d_16



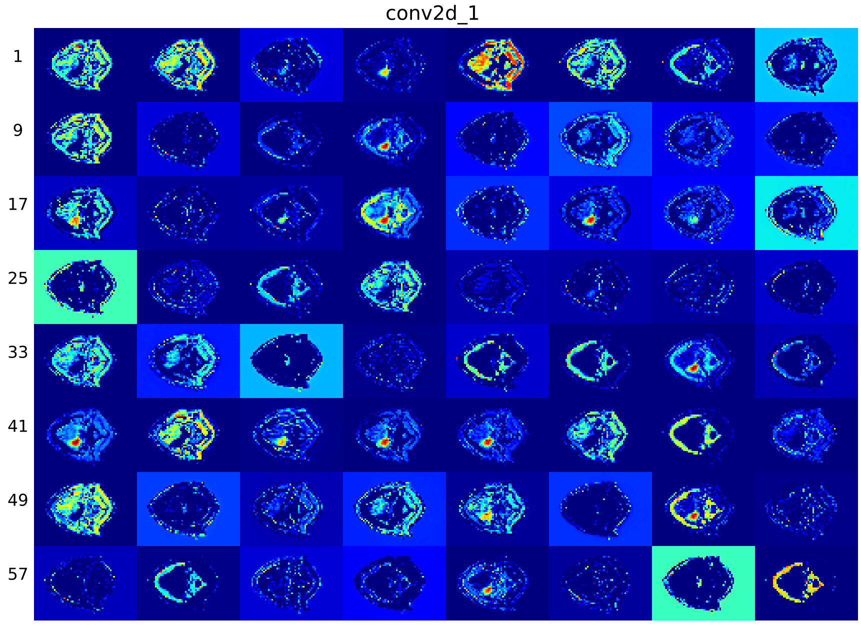
(r) Layer conv2d_17

Figure E.2: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 148, slice 11, with a Dice score of 0.94.

E.3 Activation maps patient 209, slice 14

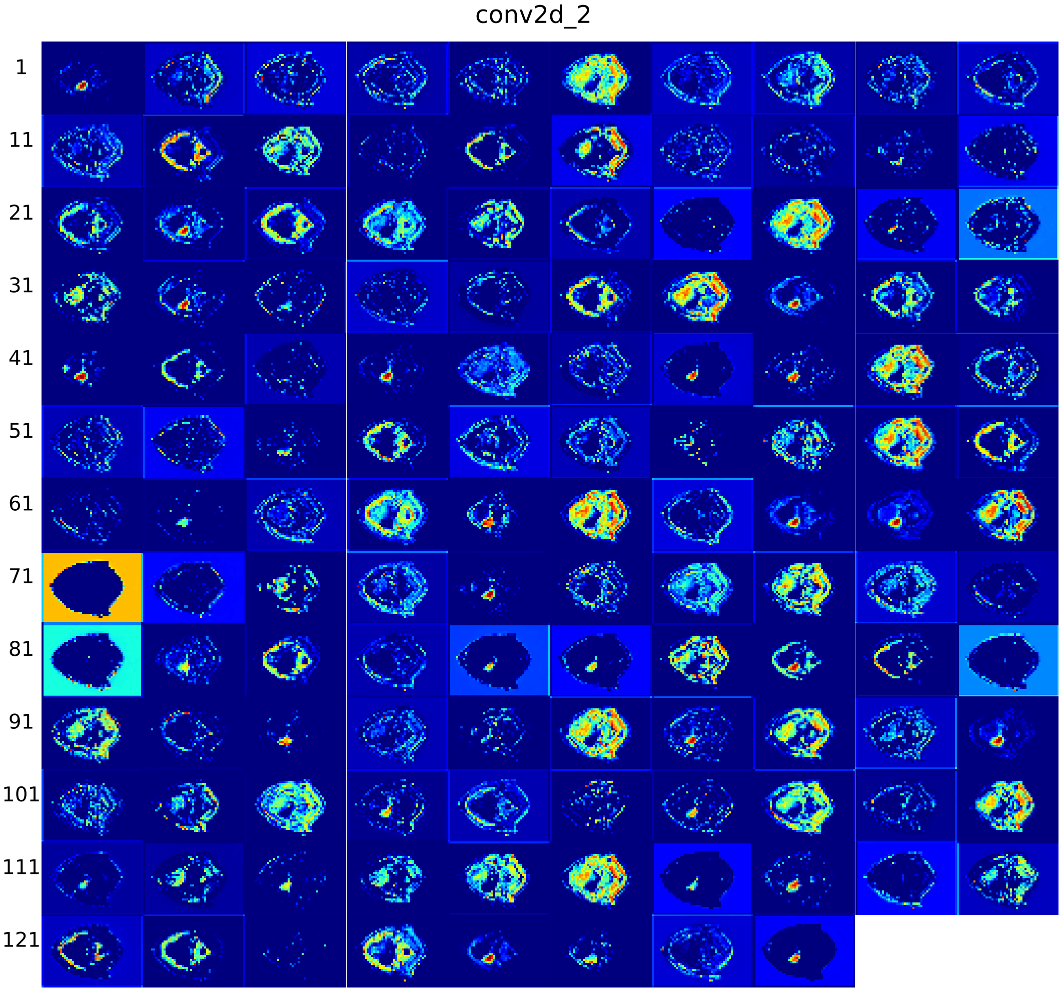


(a) Layer conv2d



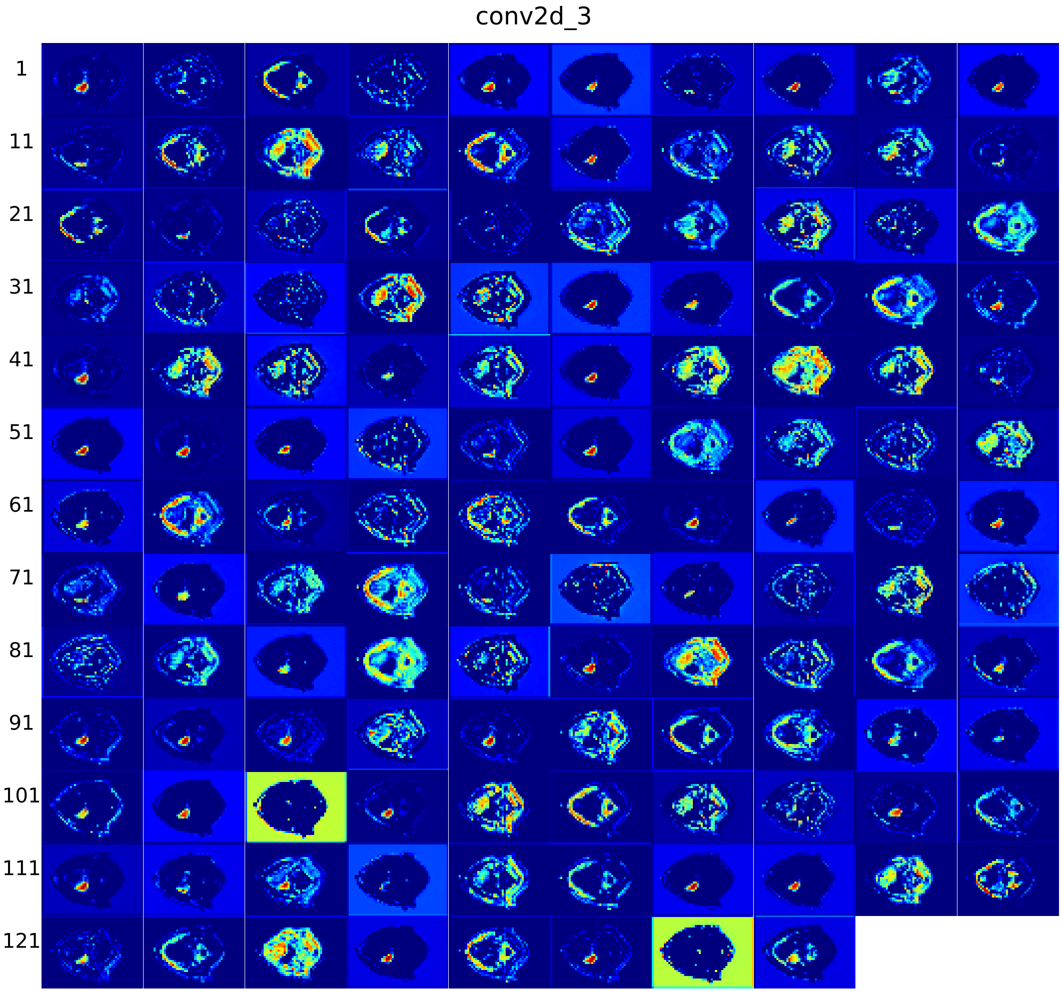
(b) Layer conv2d_1

Figure E.3: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



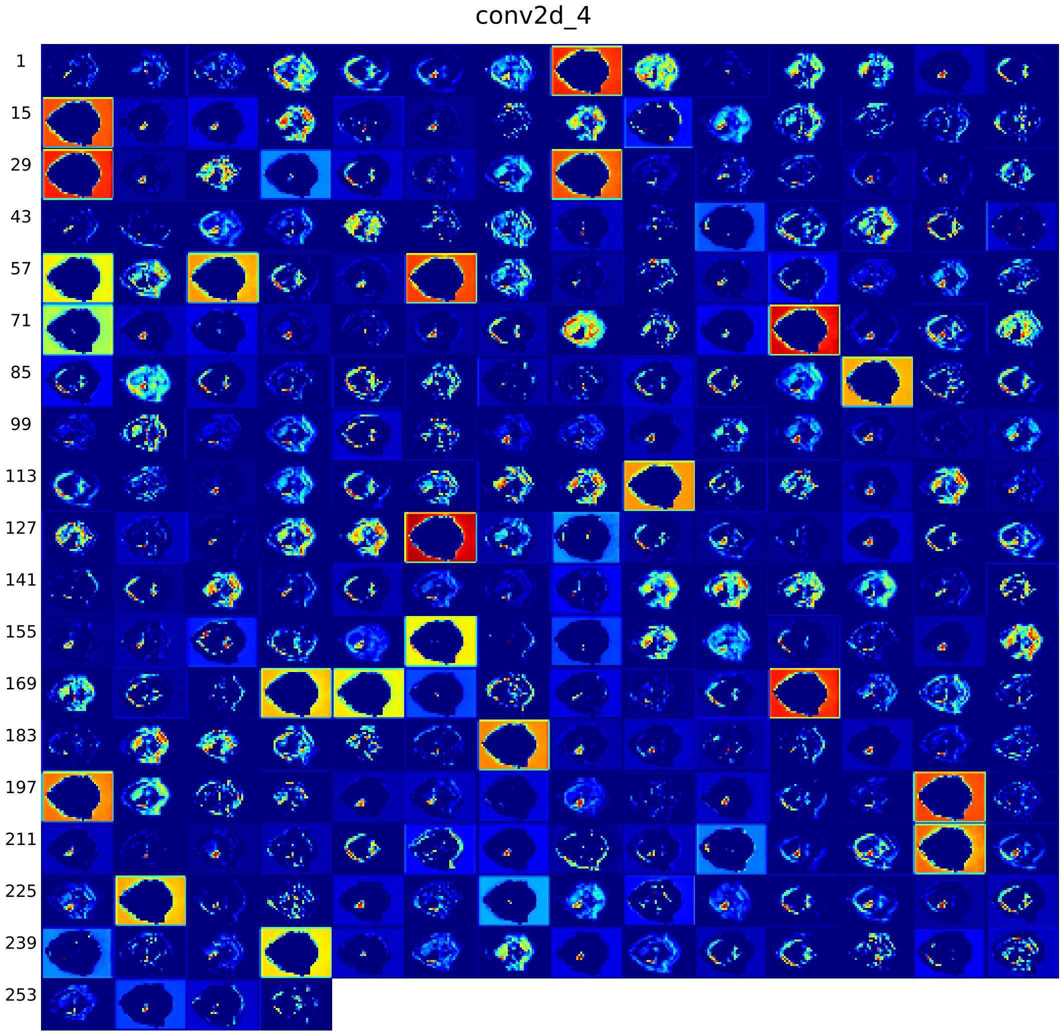
(c) Layer conv2d_2

Figure E.3: Activation maps at conv2d_2 layer (c) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



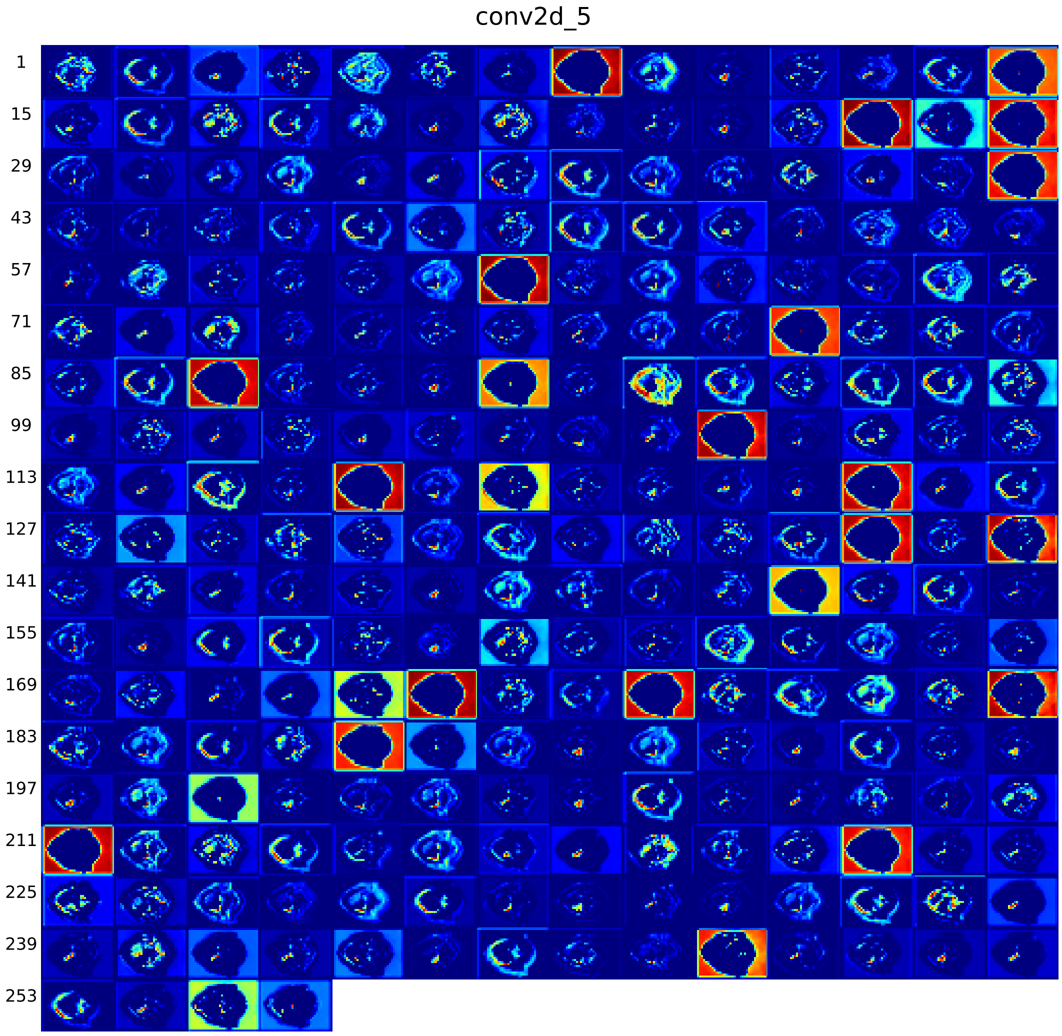
(d) Layer conv2d_3

Figure E.3: Activation maps at conv2d_3 layer (d) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



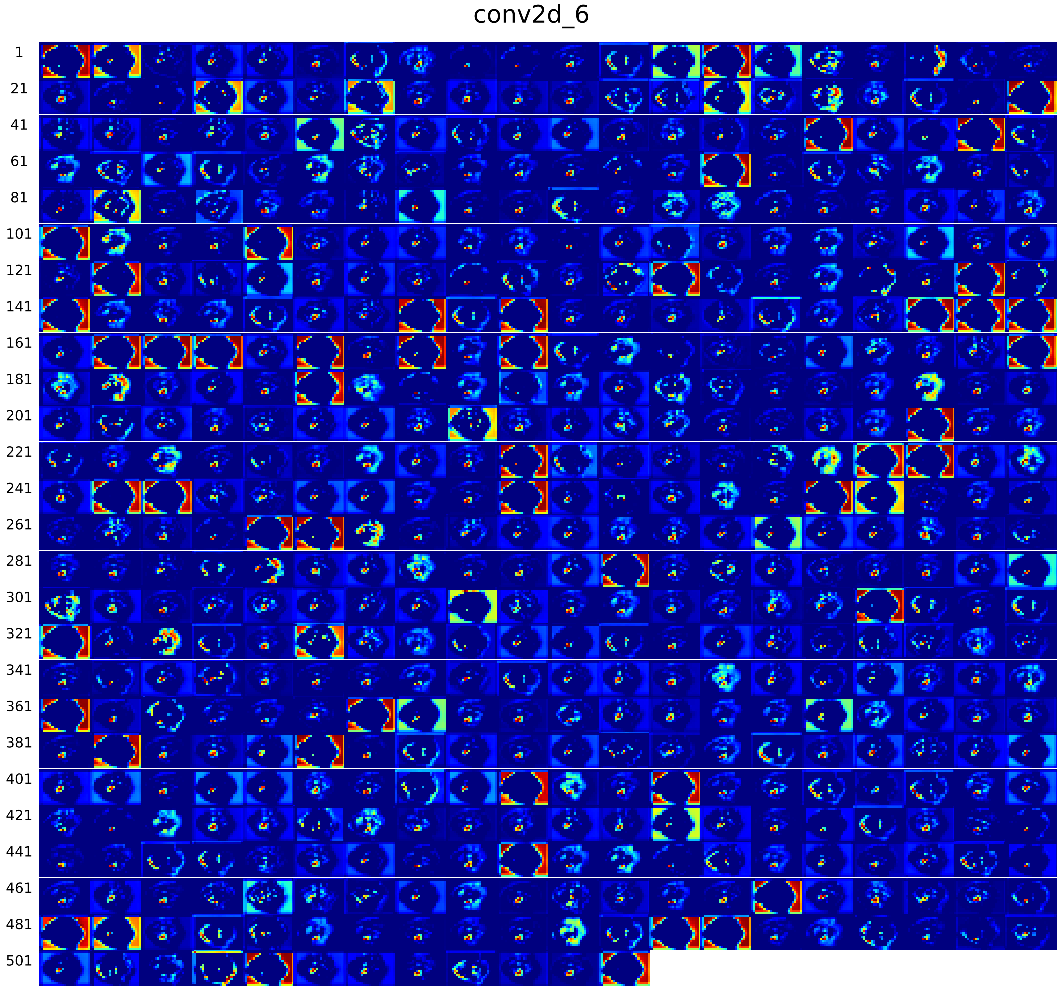
(e) Layer conv2d_4

Figure E.3: Activation maps at conv2d_4 layer (e) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



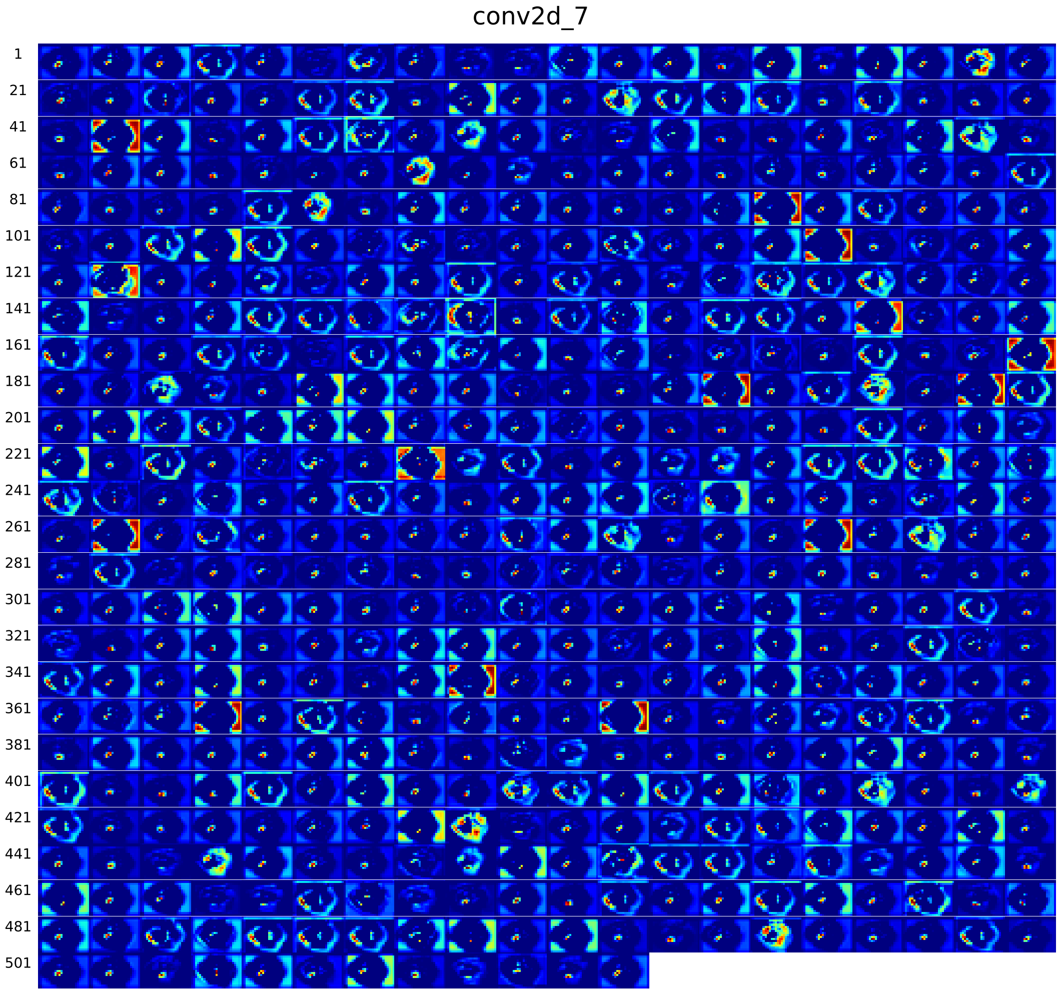
(f) Layer conv2d_5

Figure E.3: Activation maps at conv2d_5 layer (f) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page..



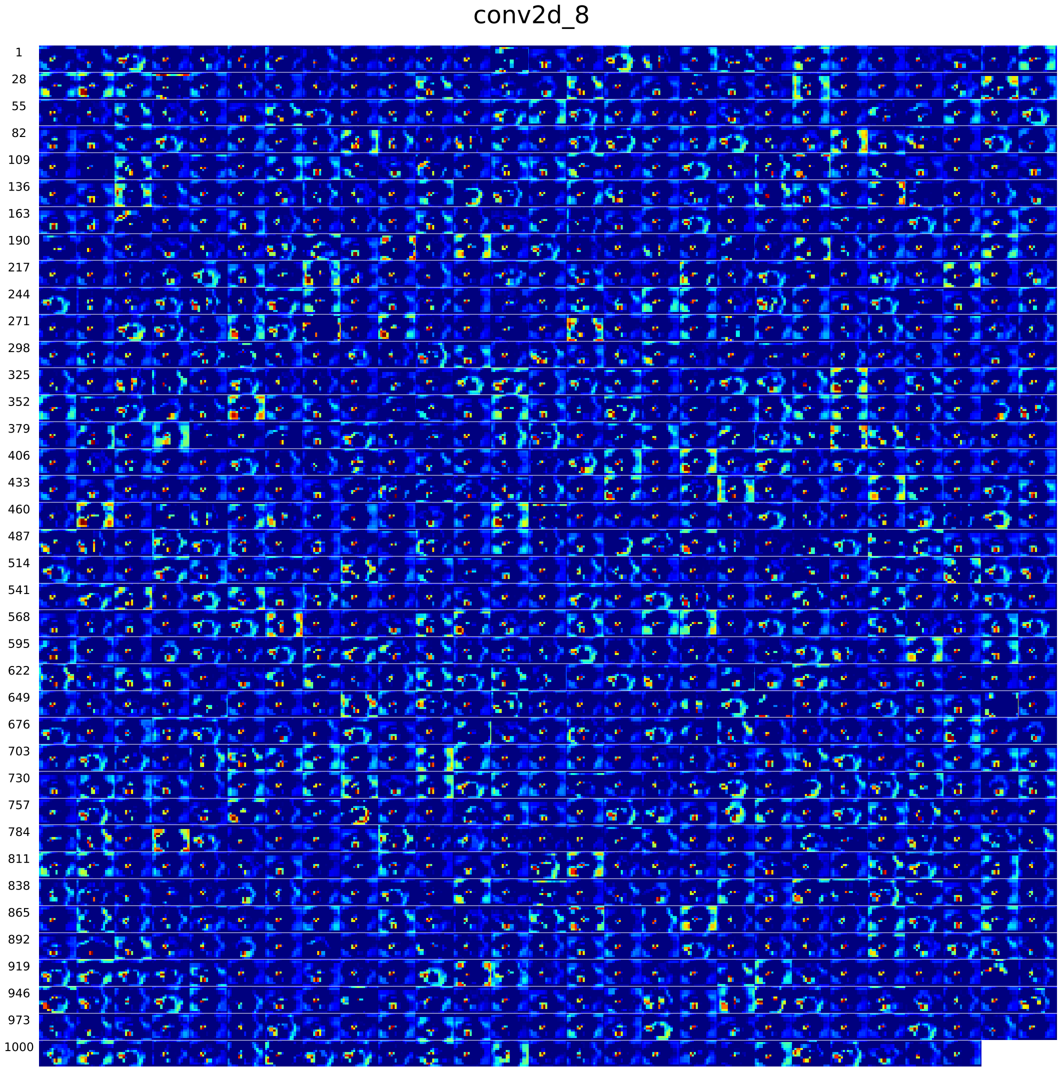
(g) Layer conv2d_6

Figure E.3: Activation maps at conv2d_6 layer (g) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



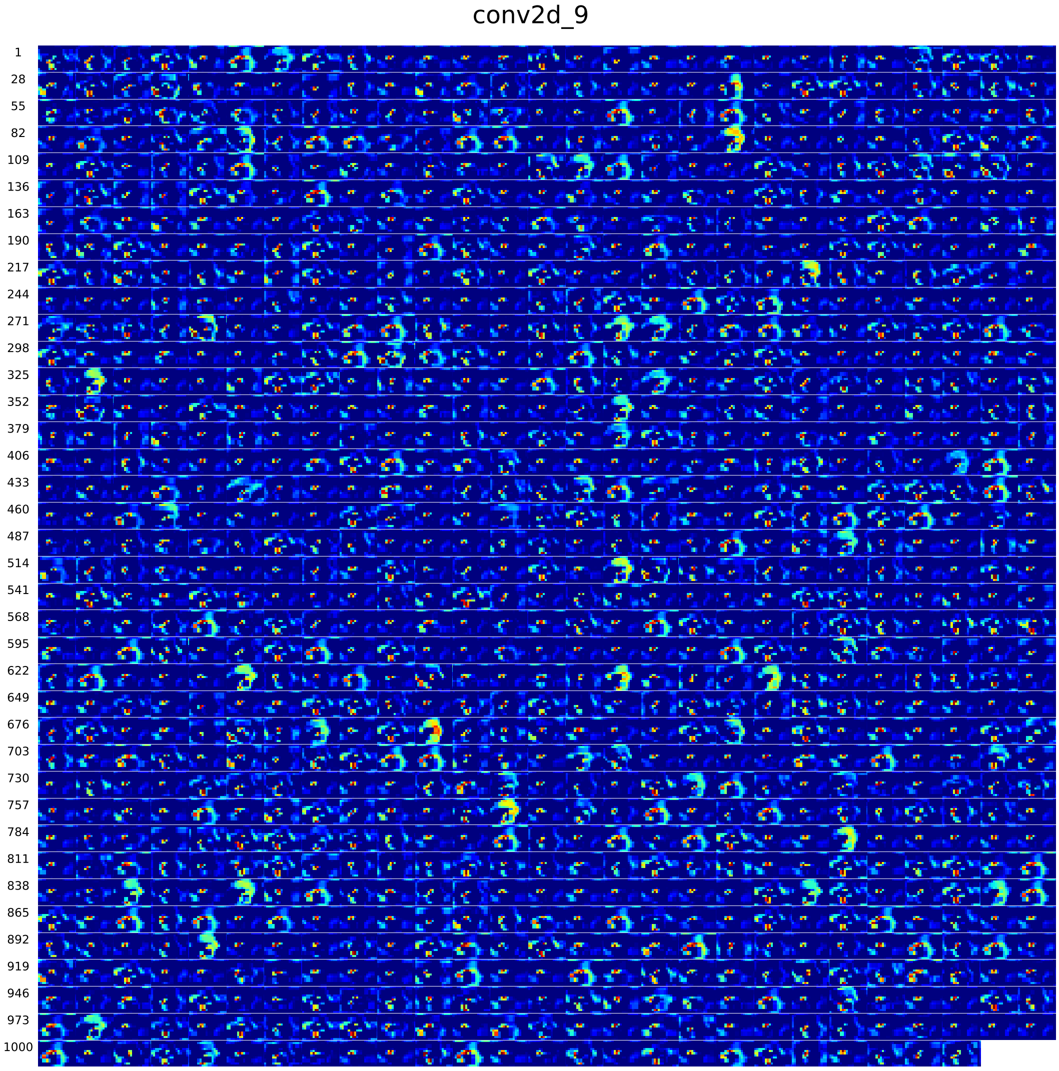
(h) Layer conv2d_7

Figure E.3: Activation maps at conv2d_7 layer (h) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



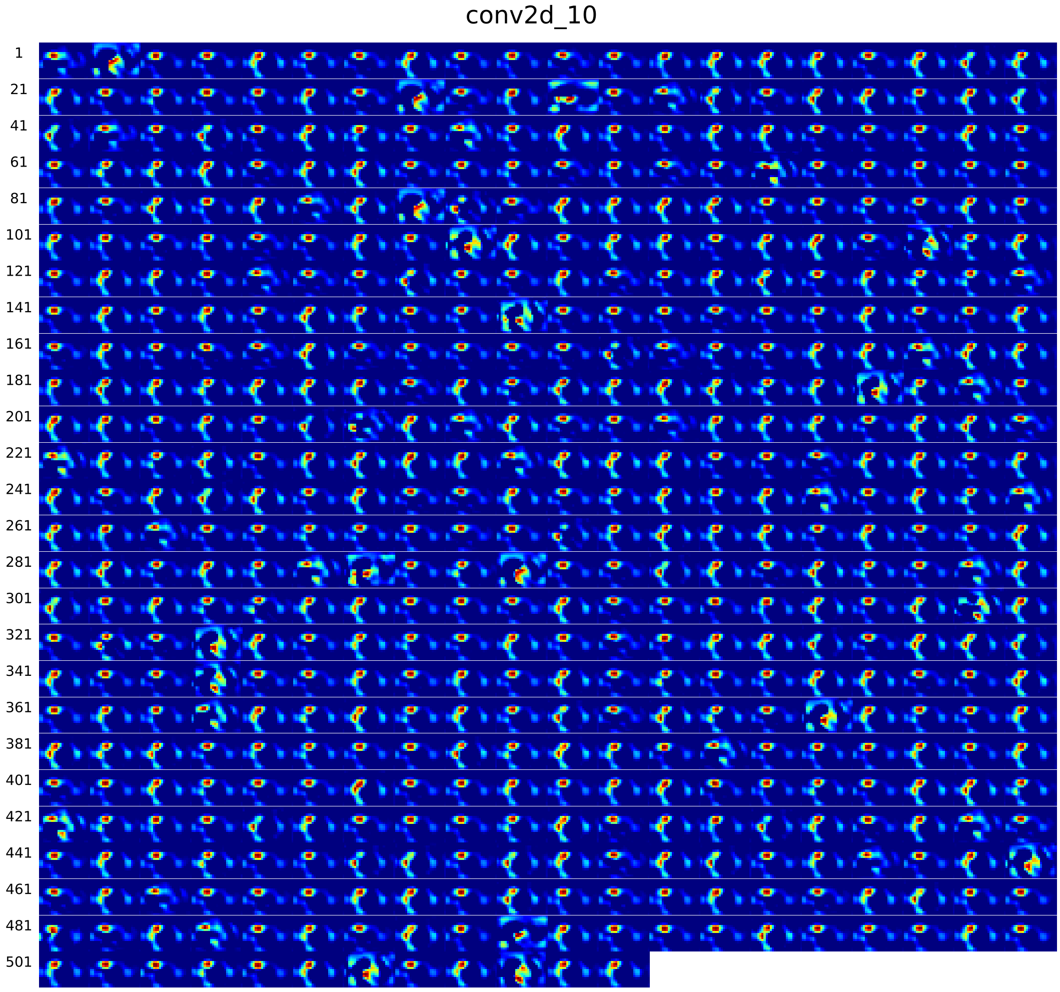
(i) Layer conv2d_8

Figure E.3: Activation maps at conv2d_8 layer (i) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



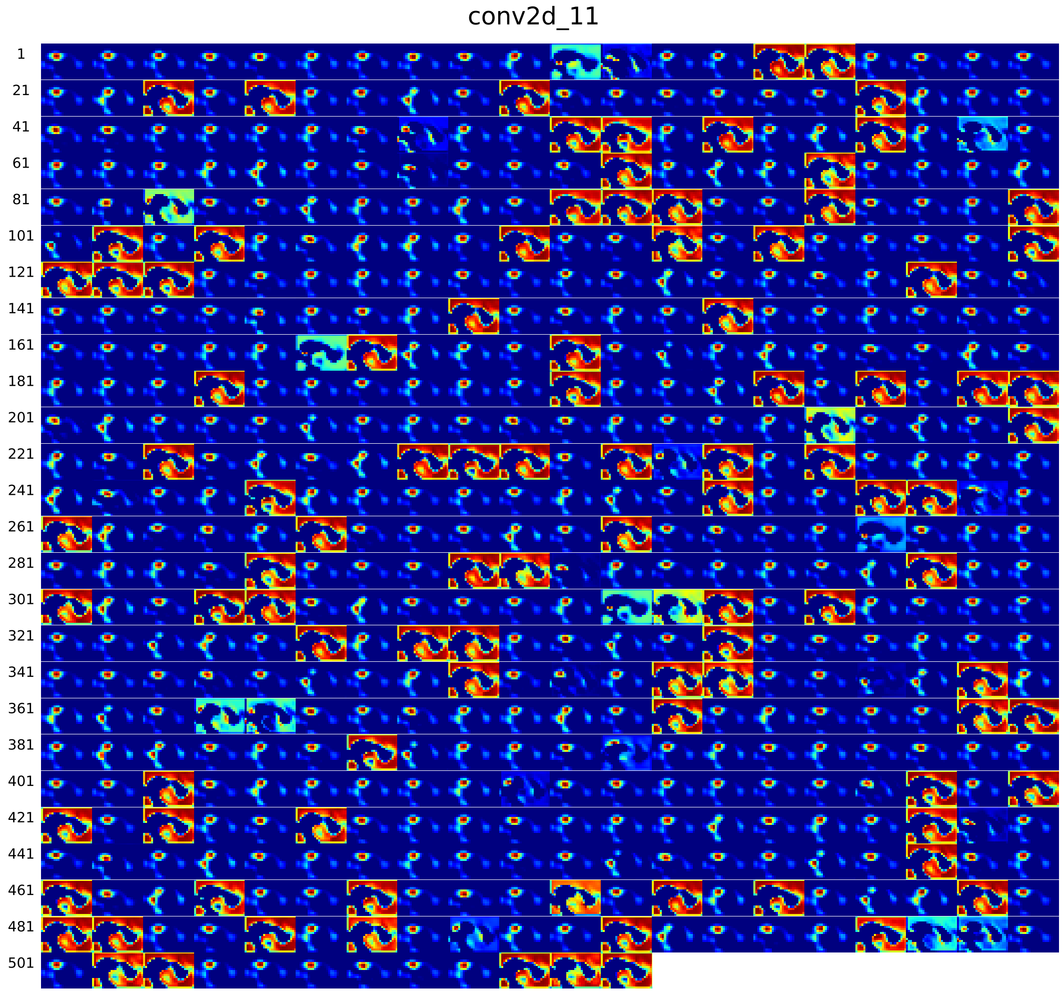
(j) Layer conv2d_9

Figure E.3: Activation maps at conv2d_9 layer (j) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



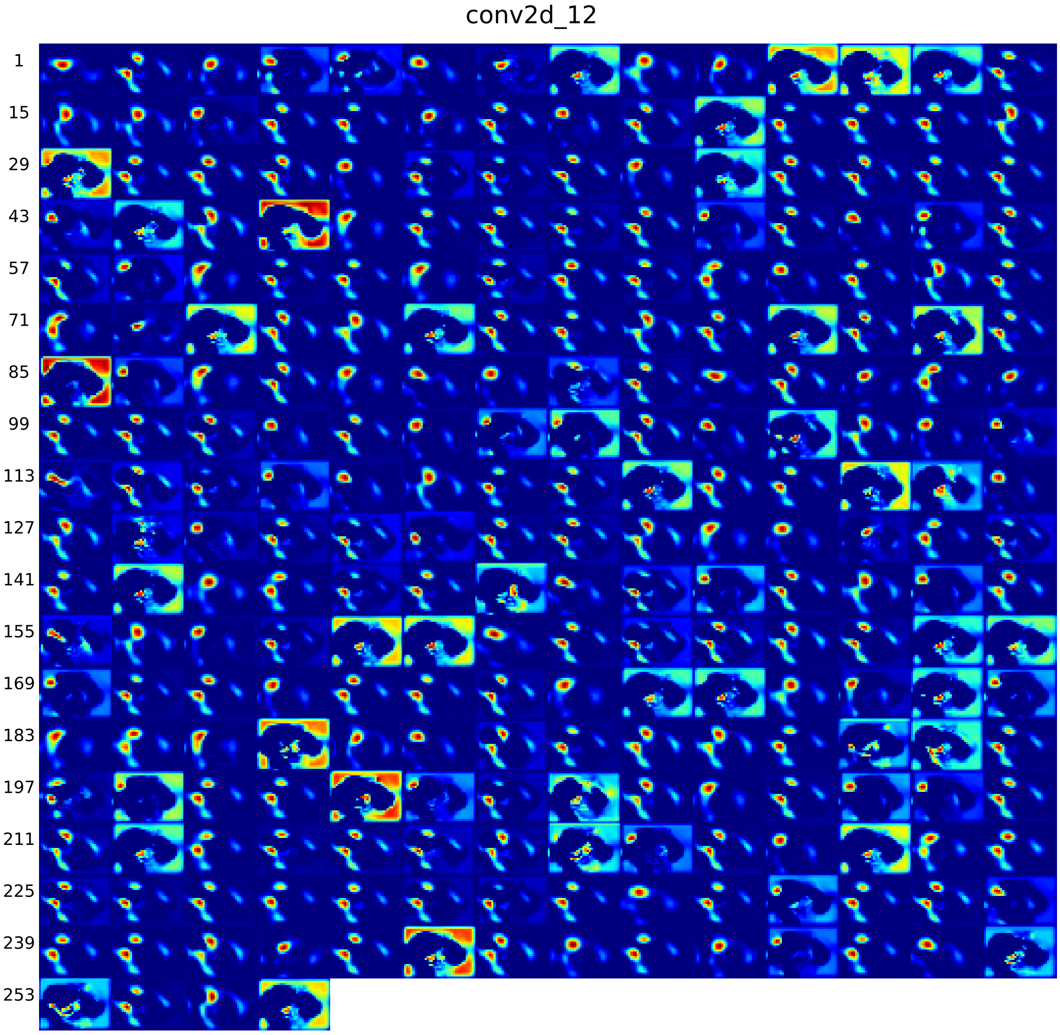
(k) Layer conv2d_10

Figure E.3: Activation maps at conv2d_10 layer (k) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



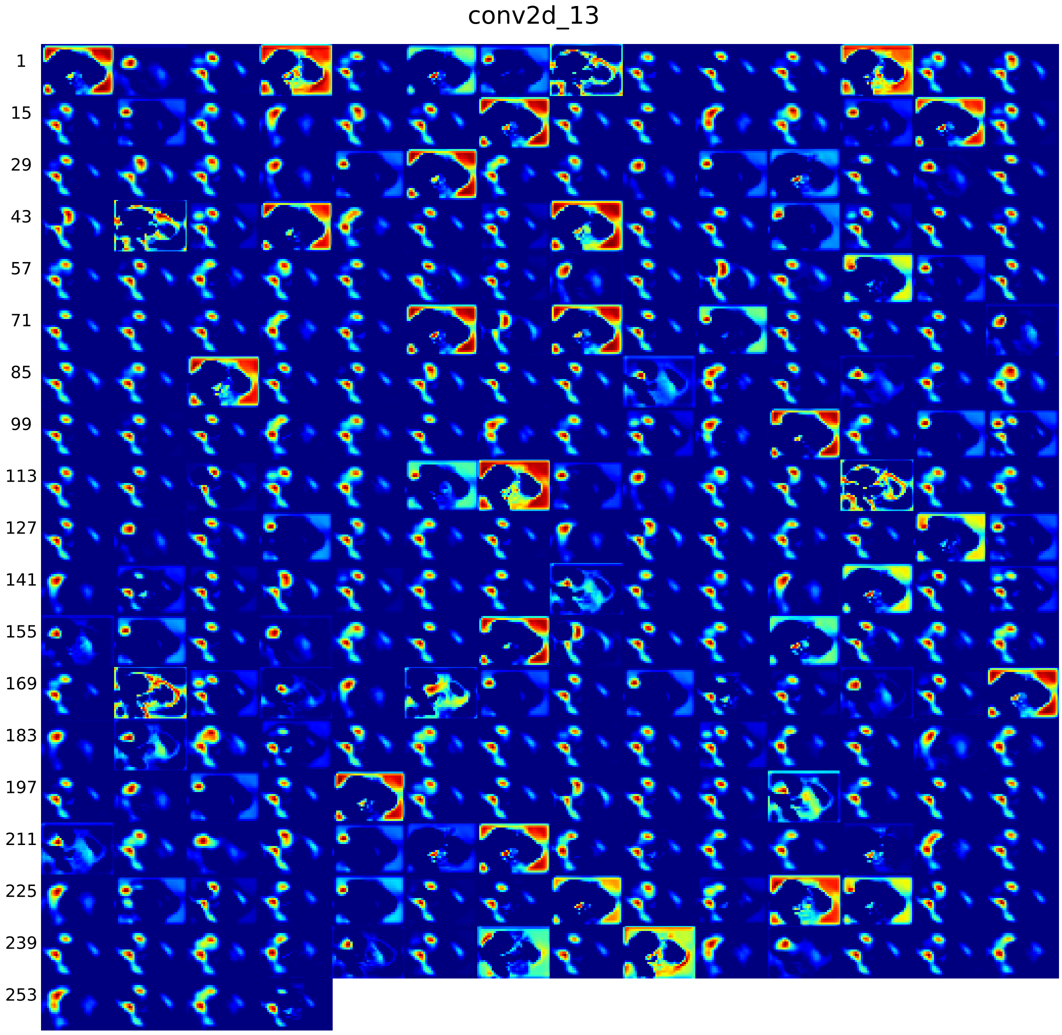
(l) Layer conv2d_11

Figure E.3: Activation maps at layer of image from patient 209, slice 14, with a Dice score of 0.94. (l) Layer conv2d_14. Continued on next page..



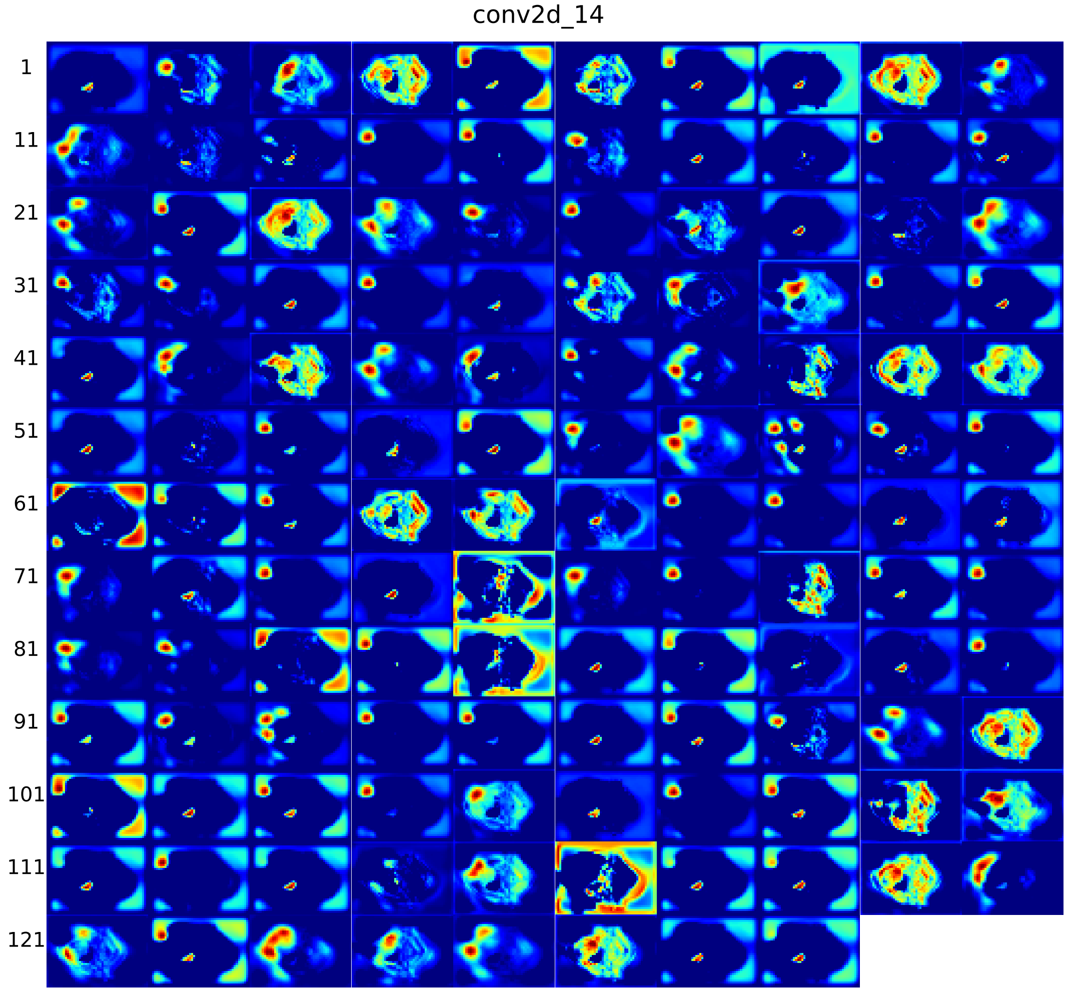
(m) Layer conv2d_12

Figure E.3: Activation maps at conv2d_12 layer (m) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



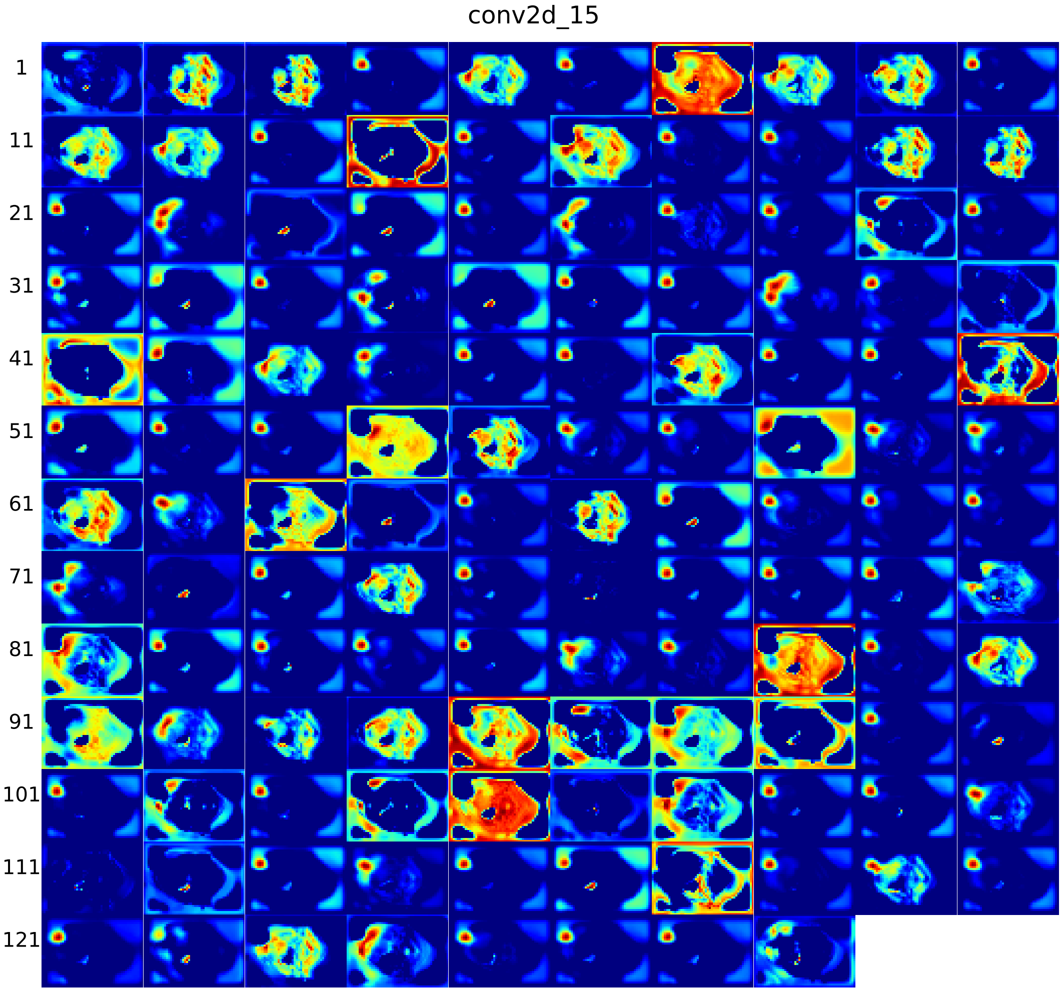
(n) Layer conv2d_13

Figure E.3: Activation maps at conv2d_13 layer (n) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.



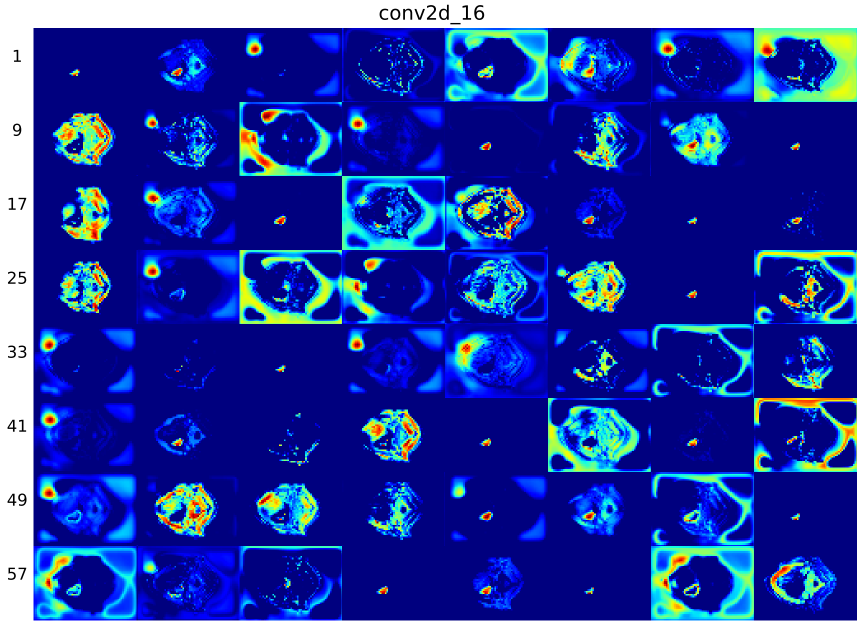
(o) Layer conv2d_14

Figure E.3: Activation maps at conv2d_14 layer (o) of image from patient 209, slice 14, with a Dice score of 0.94. Continued on next page.

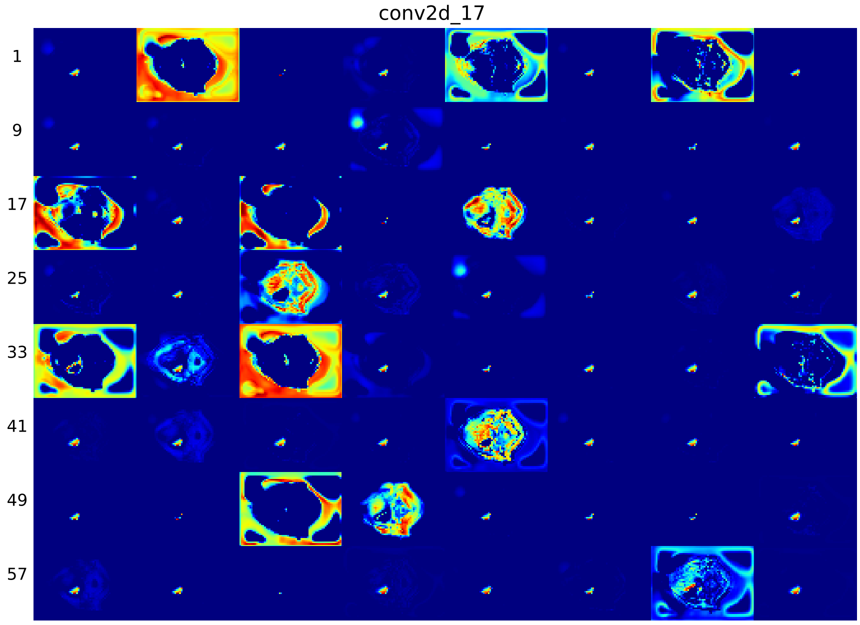


(p) Layer conv2d_15

Figure E.3: Activation maps at layer of image from patient 209, slice 14, with a Dice score of 0.94. (p) Layer conv2d_14. Continued on next page.



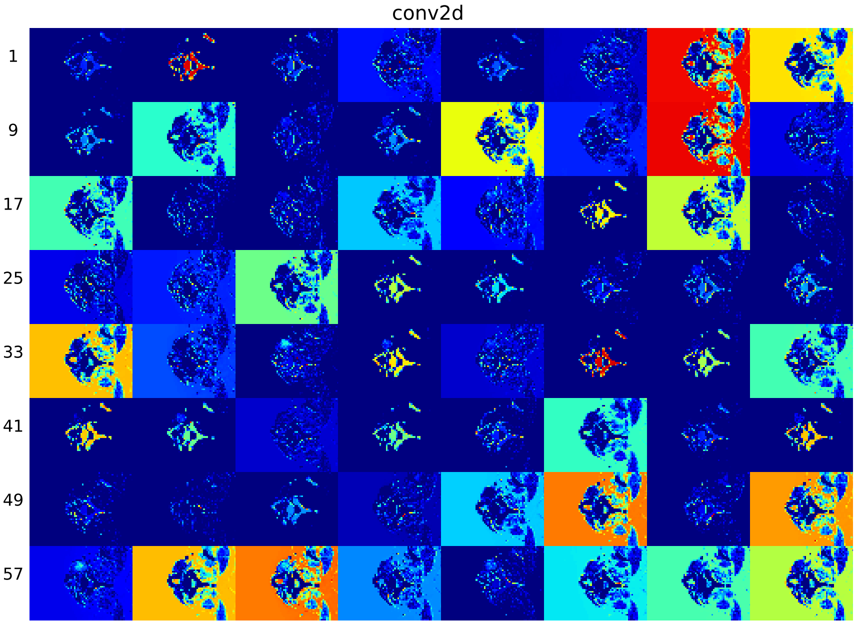
(q) Layer conv2d_16



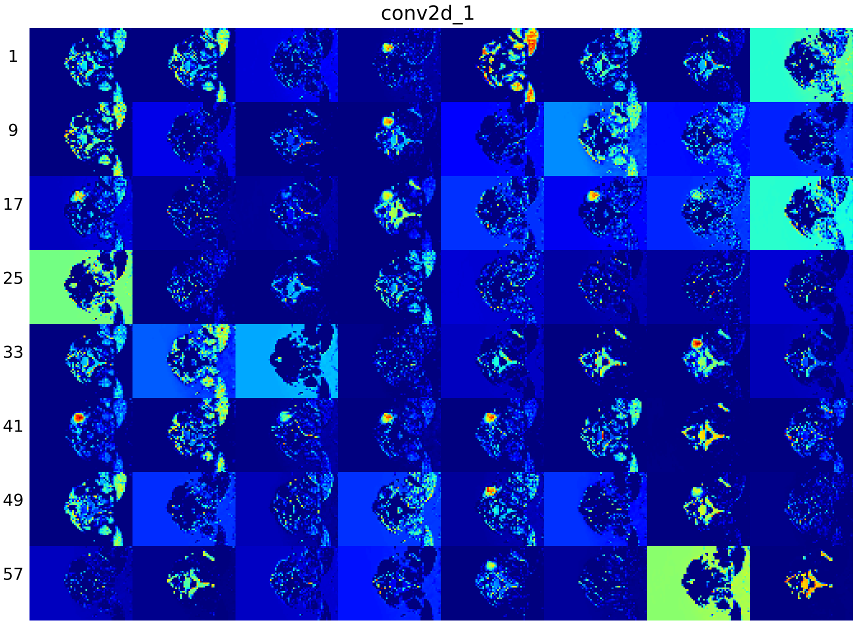
(r) Layer conv2d_17

Figure E.3: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 209, slice 14, with a Dice score of 0.94.

E.4 Activation maps patient 217, slice 20

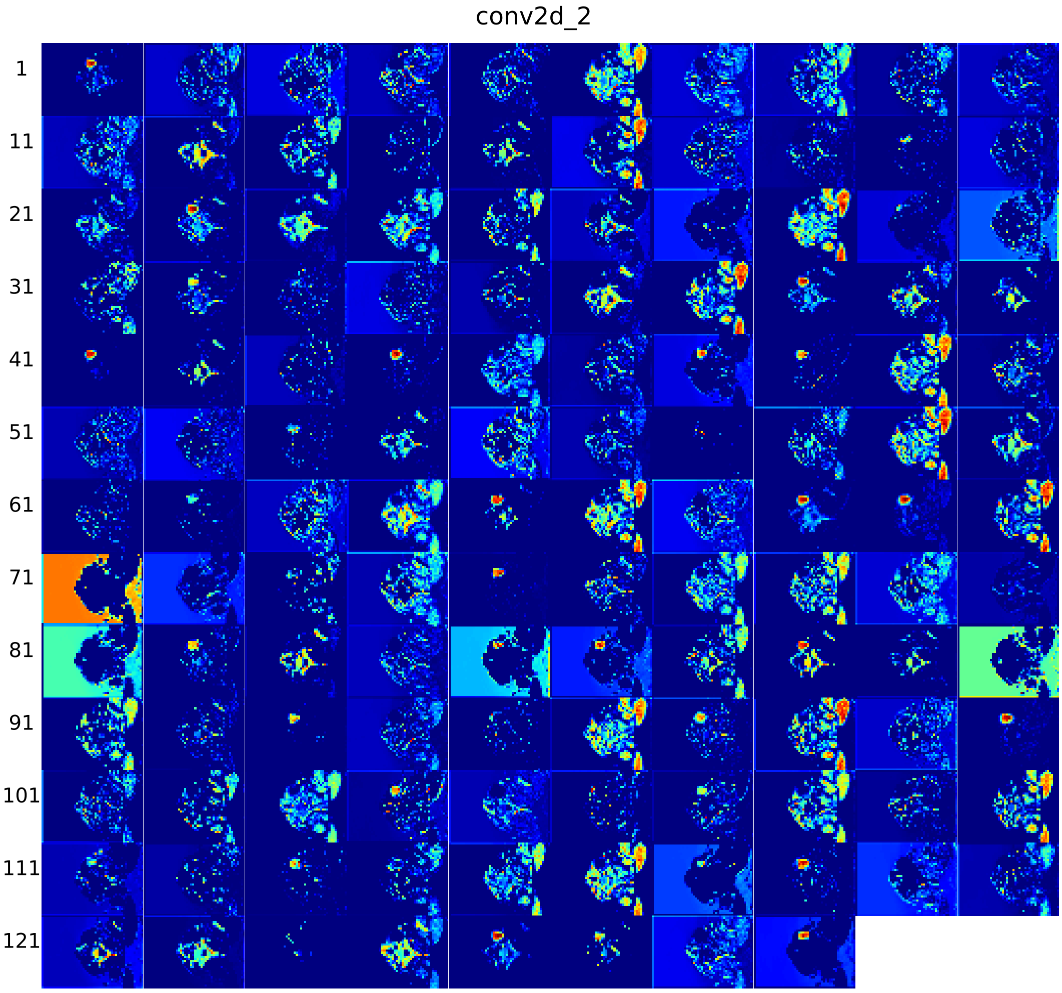


(a) Layer conv2d



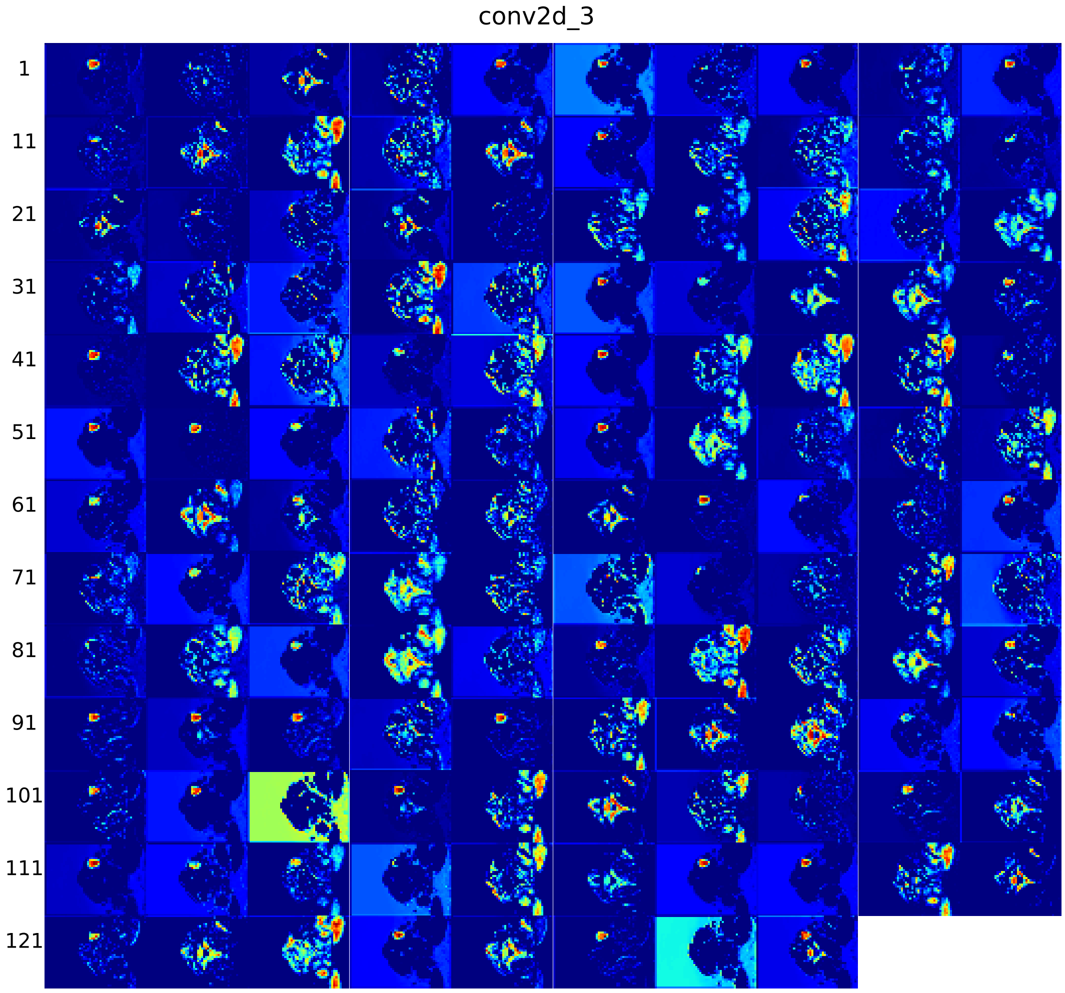
(b) Layer conv2d_1

Figure E.4: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



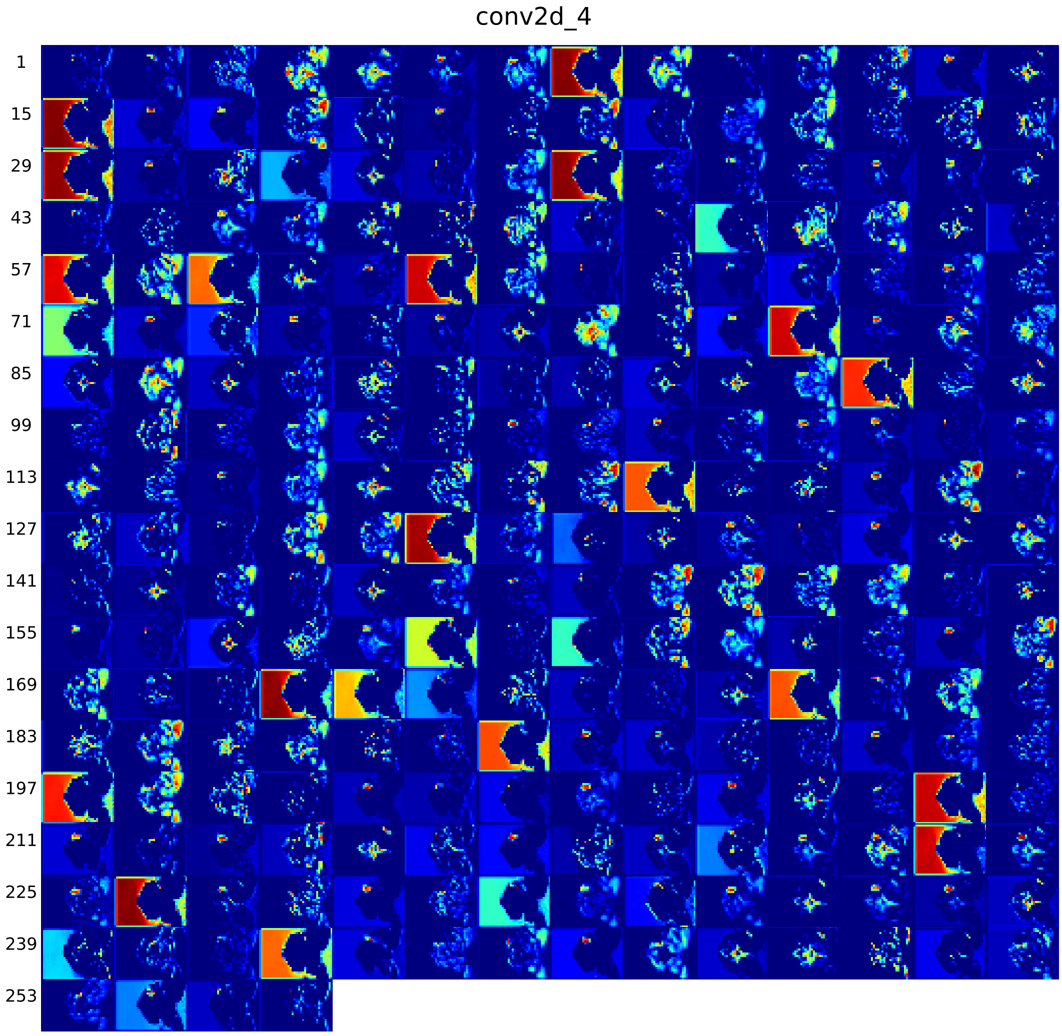
(c) Layer conv2d_2

Figure E.4: Activation maps at conv2d_2 layer (c) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



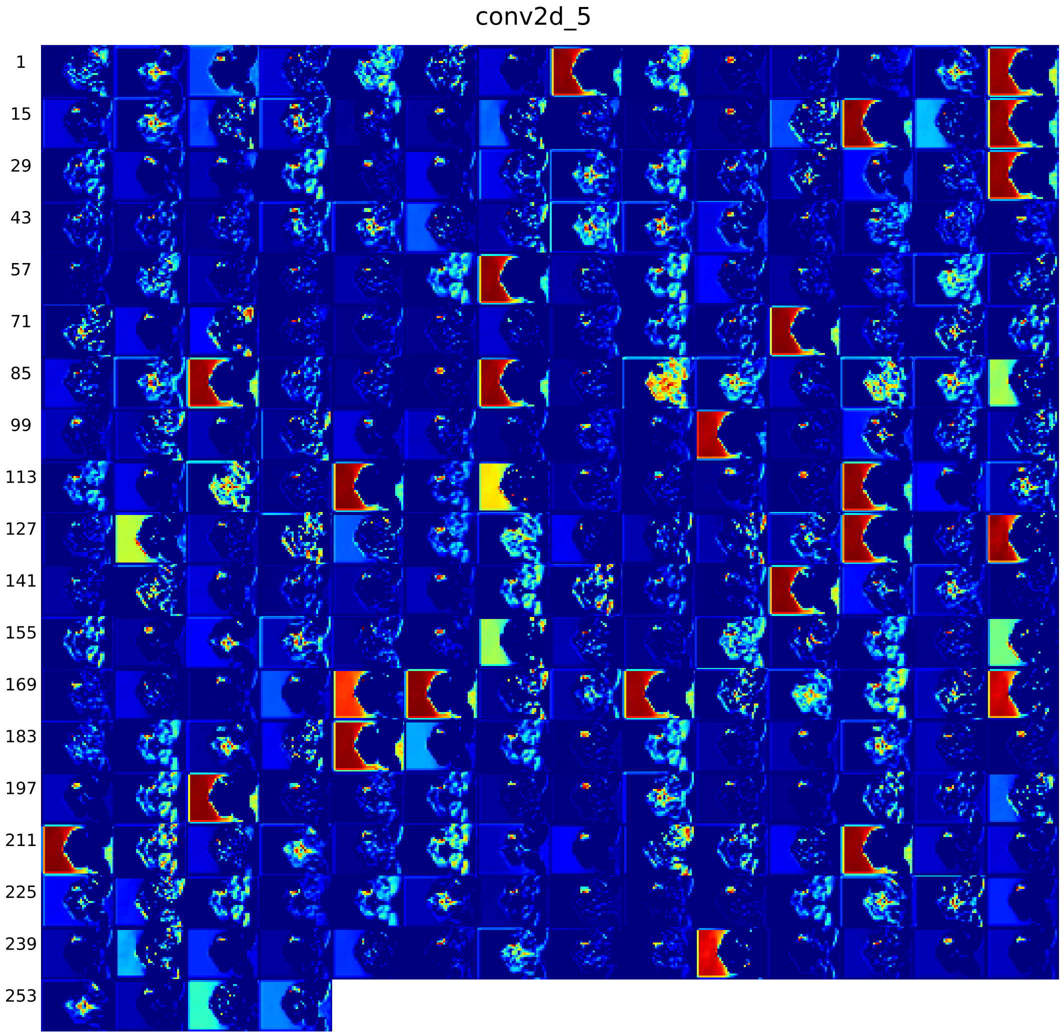
(d) Layer conv2d_3

Figure E.4: Activation maps at conv2d_3 layer (d) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



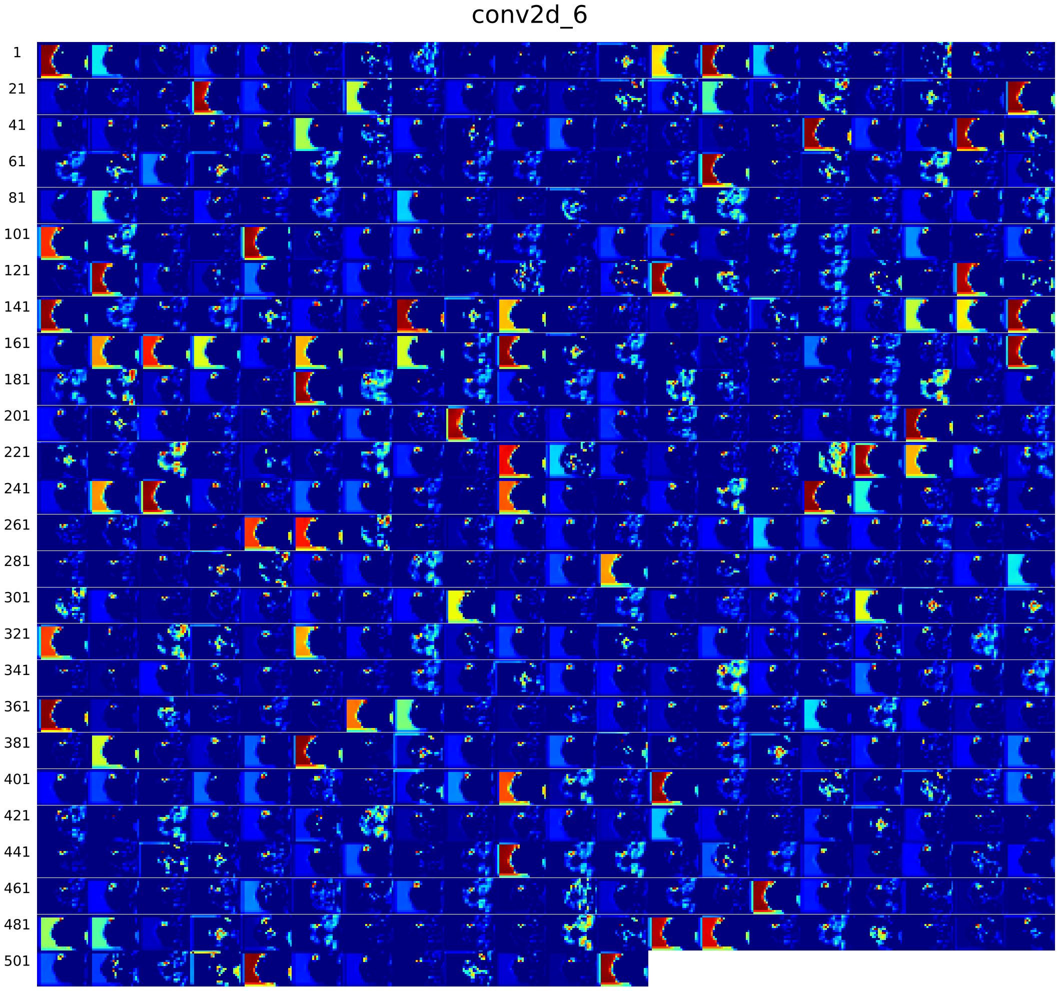
(e) Layer conv2d_4

Figure E.4: Activation maps at conv2d_4 layer (e) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



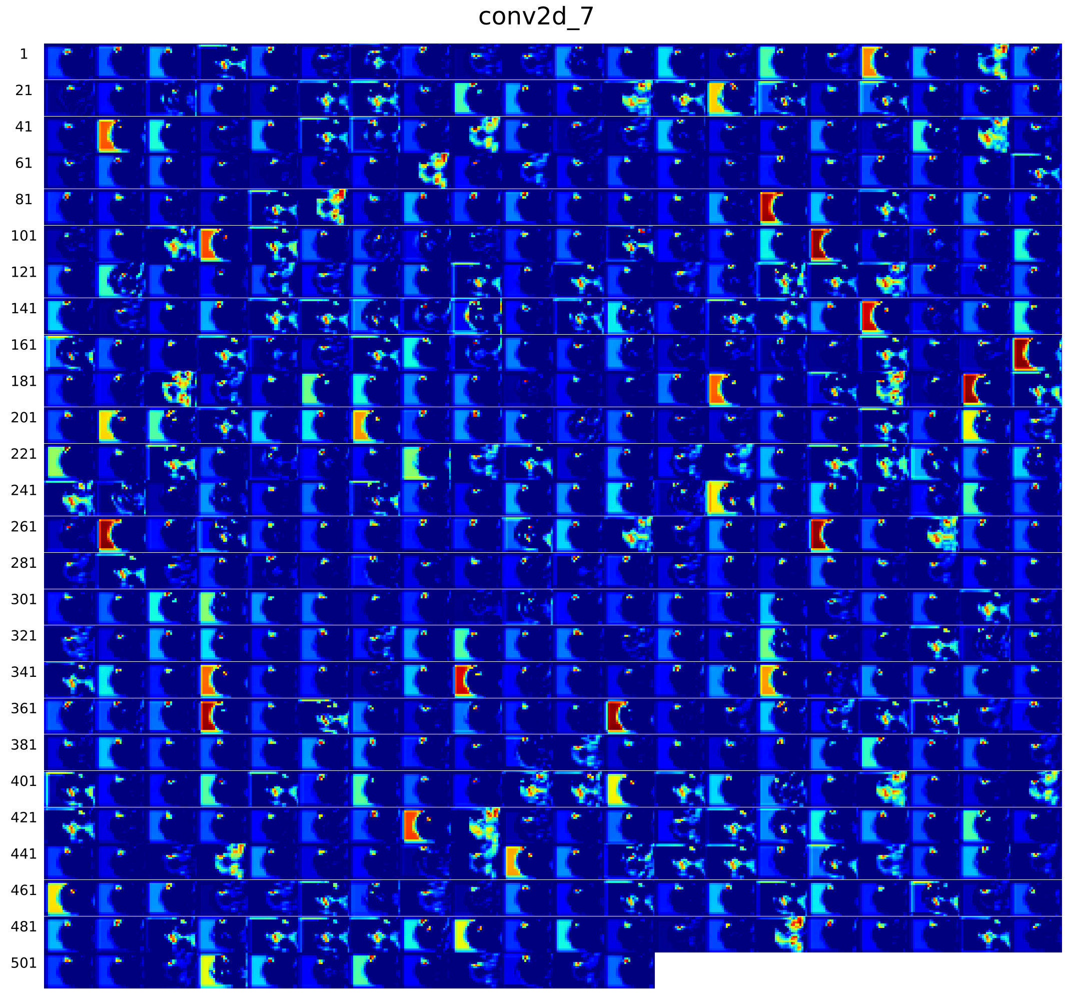
(f) Layer conv2d_5

Figure E.4: Activation maps at conv2d_5 layer (f) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



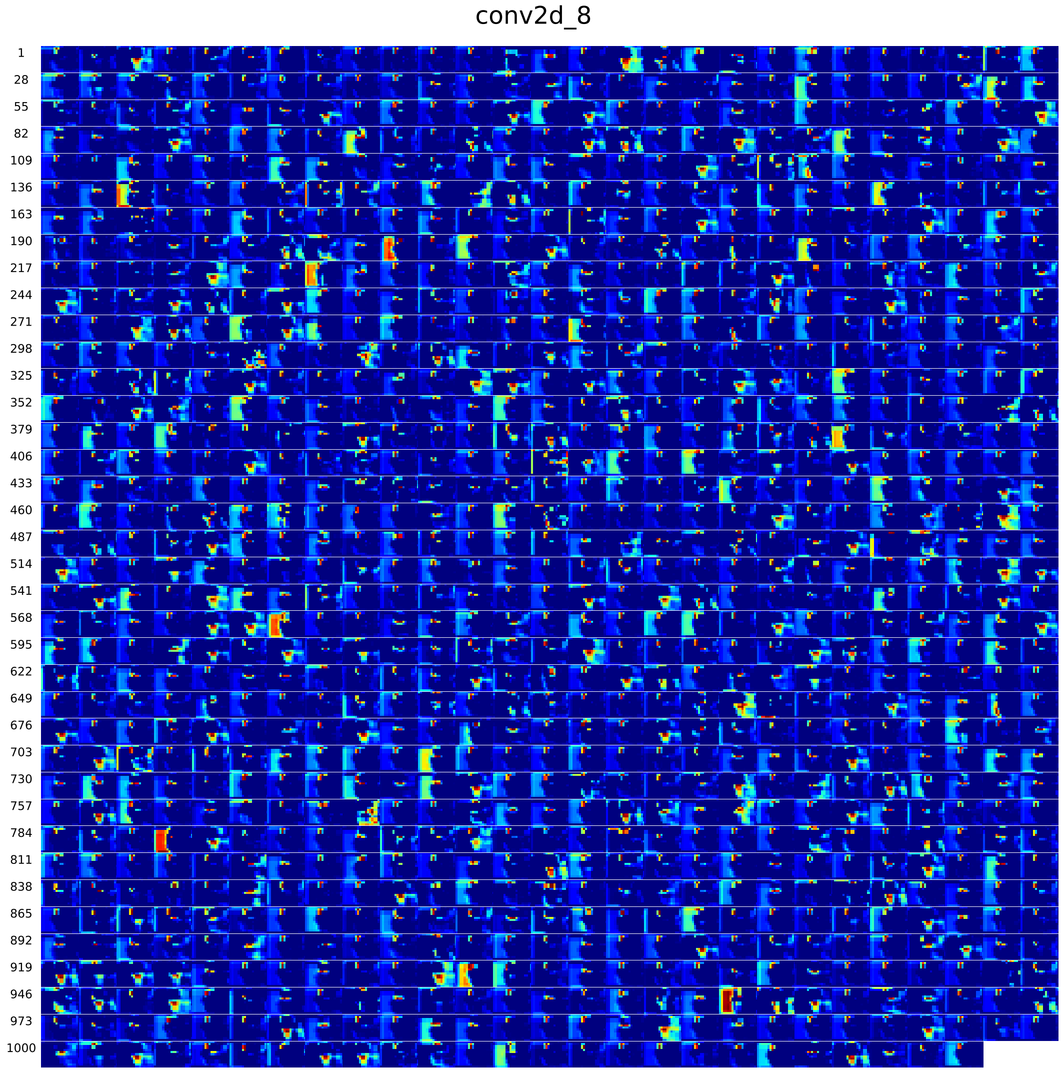
(g) Layer conv2d_6

Figure E.4: Activation maps at conv2d_6 layer (g) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



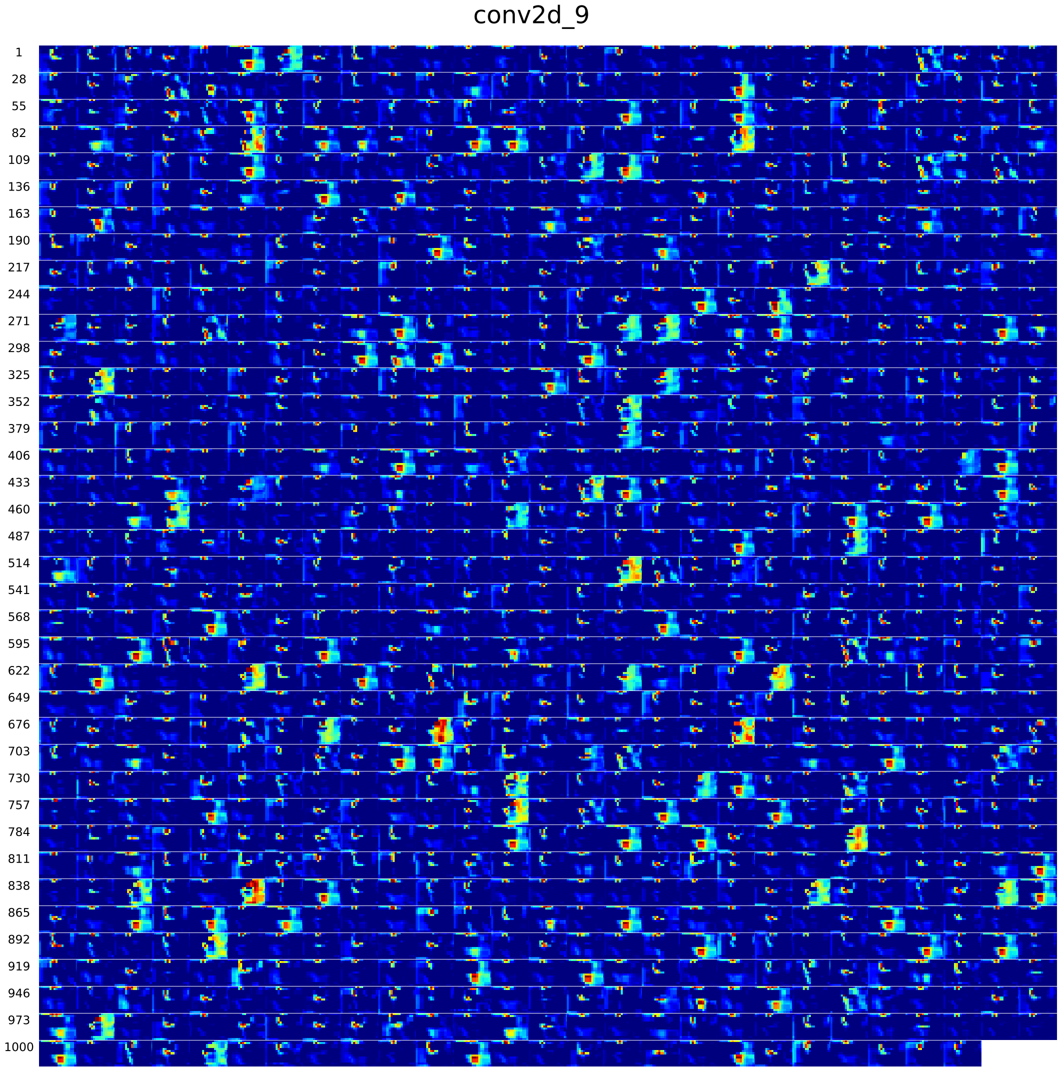
(h) Layer conv2d_7

Figure E.4: Activation maps at conv2d_7 layer (h) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



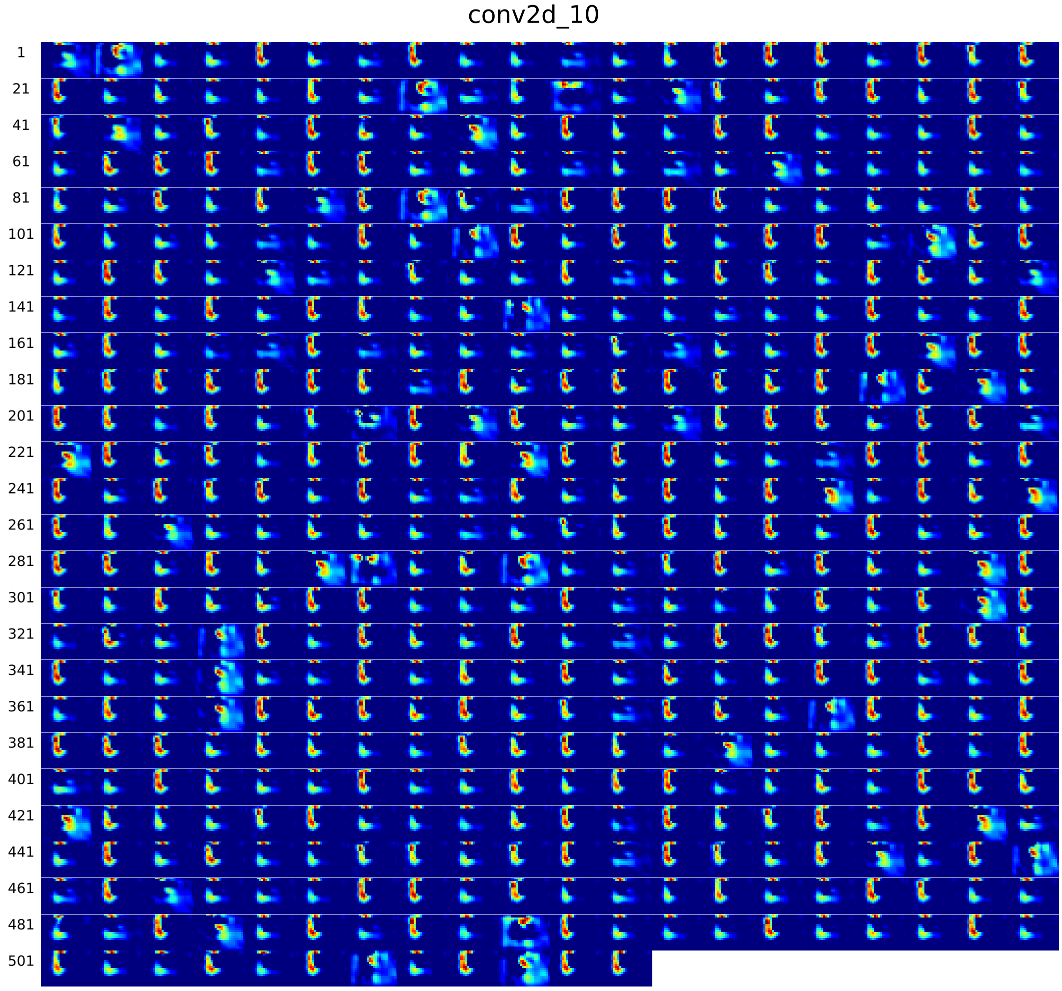
(i) Layer conv2d_8

Figure E.4: Activation maps at conv2d_8 layer (i) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



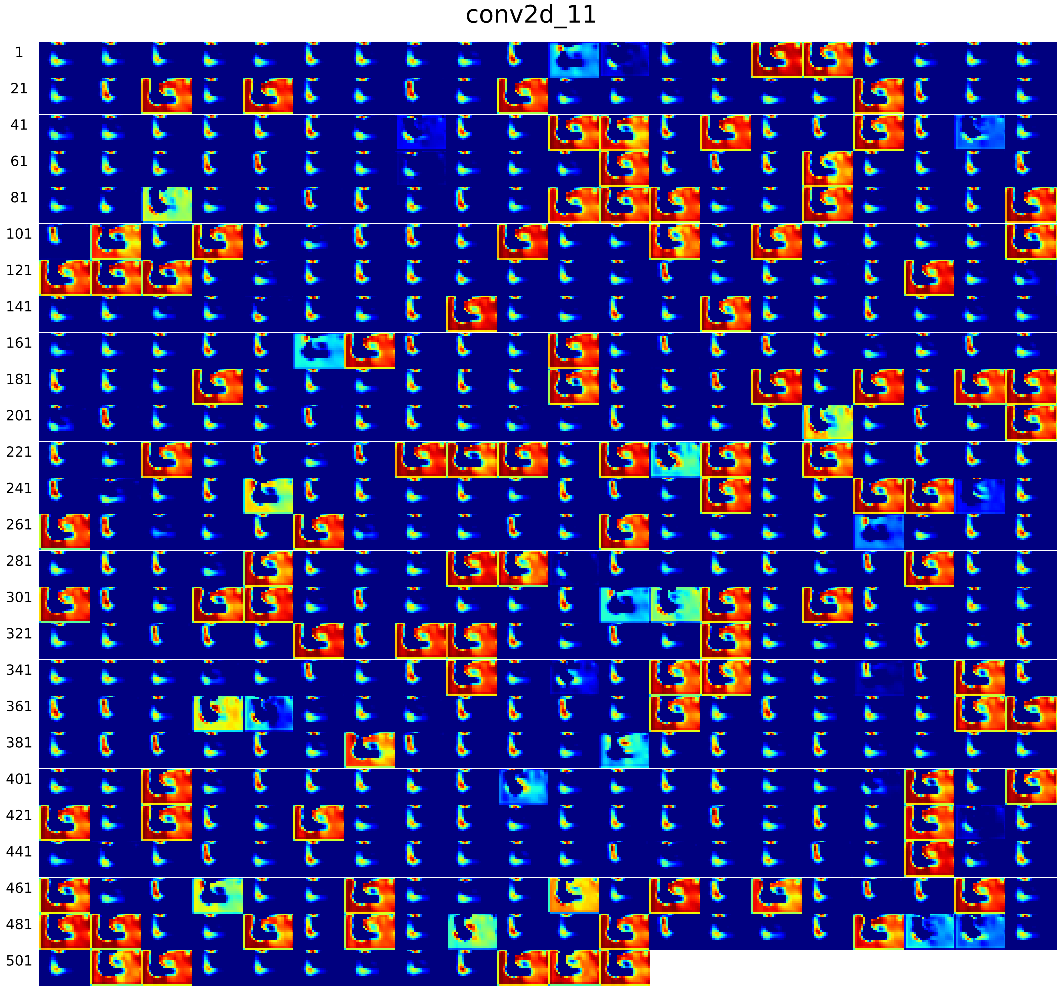
(j) Layer conv2d_9

Figure E.4: Activation maps at conv2d_9 layer (j) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



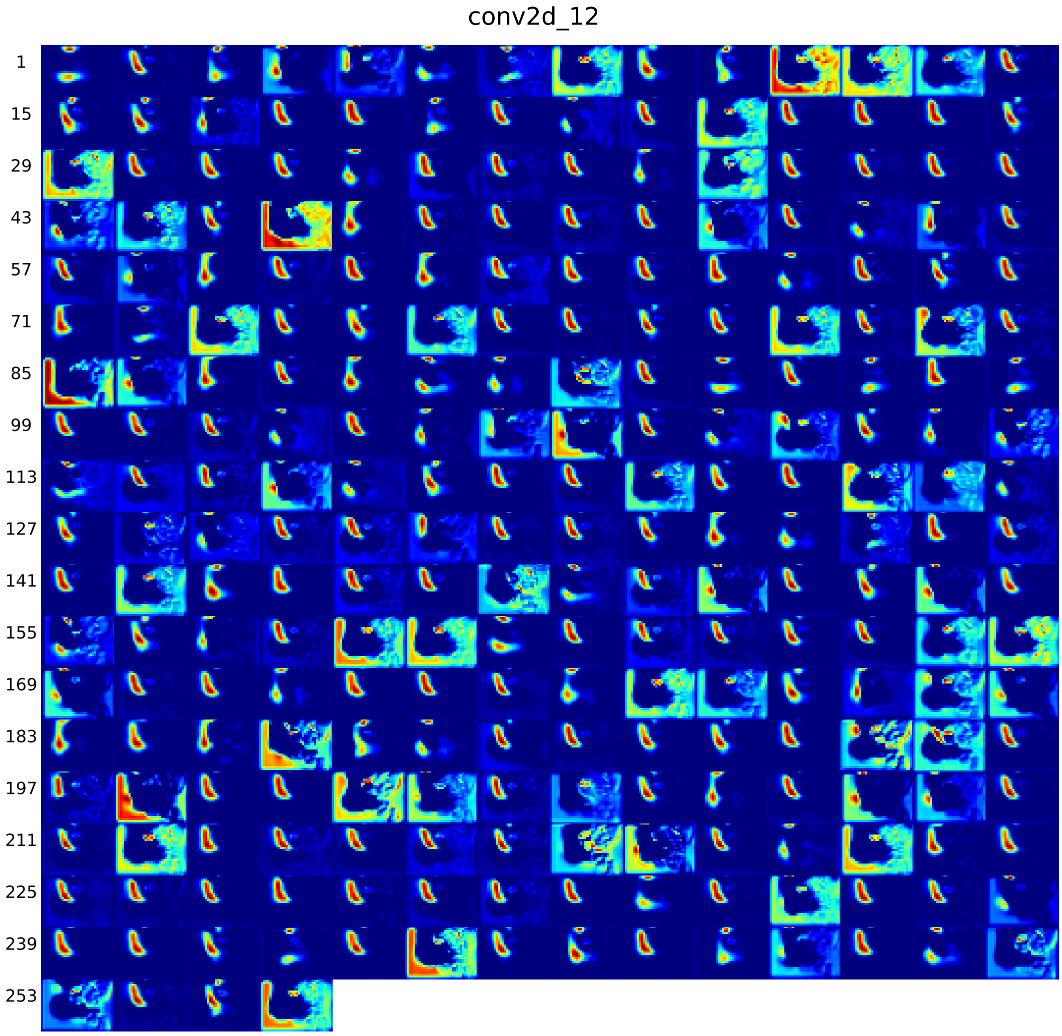
(k) Layer conv2d_10

Figure E.4: Activation maps at conv2d_10 layer (k) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



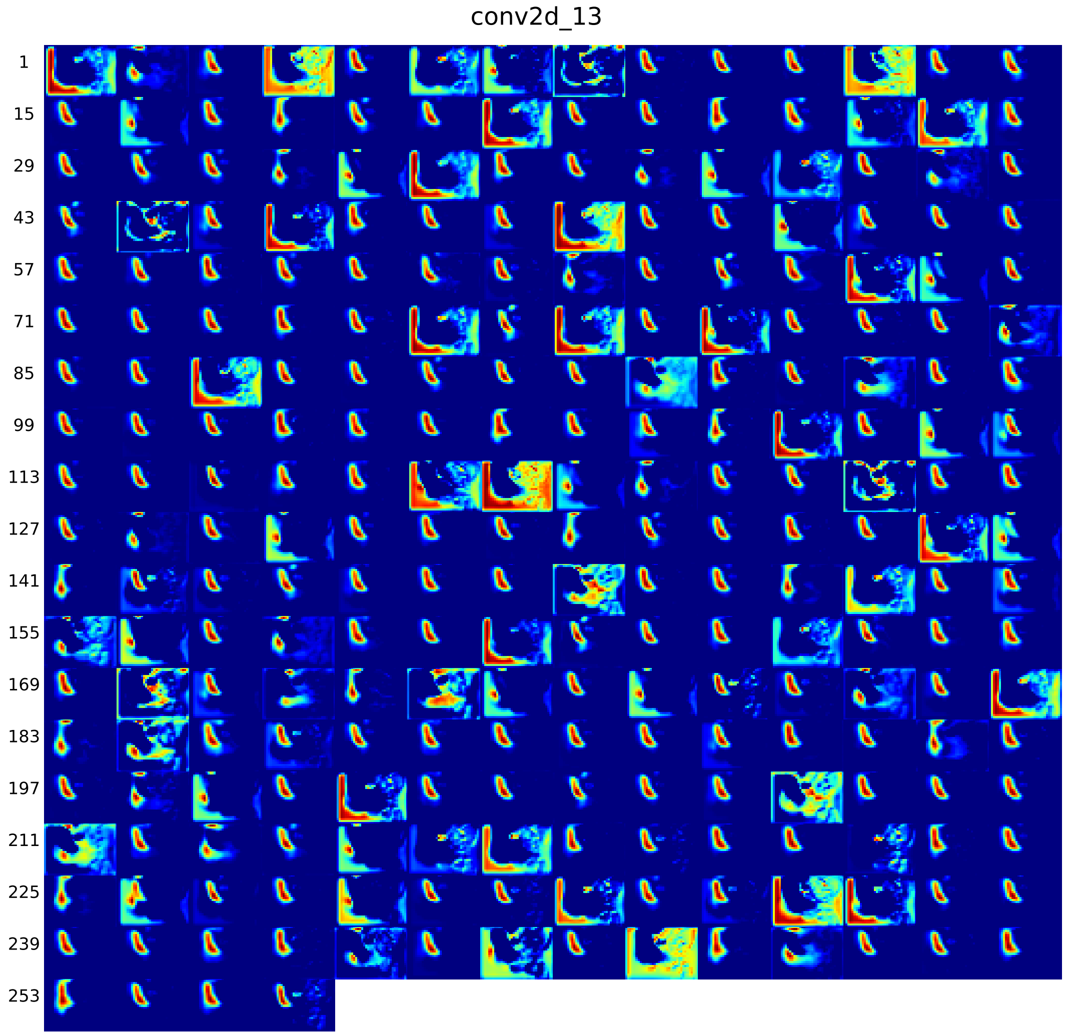
(l) Layer conv2d_11

Figure E.4: Activation maps at conv2d_11 layer (l) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page..



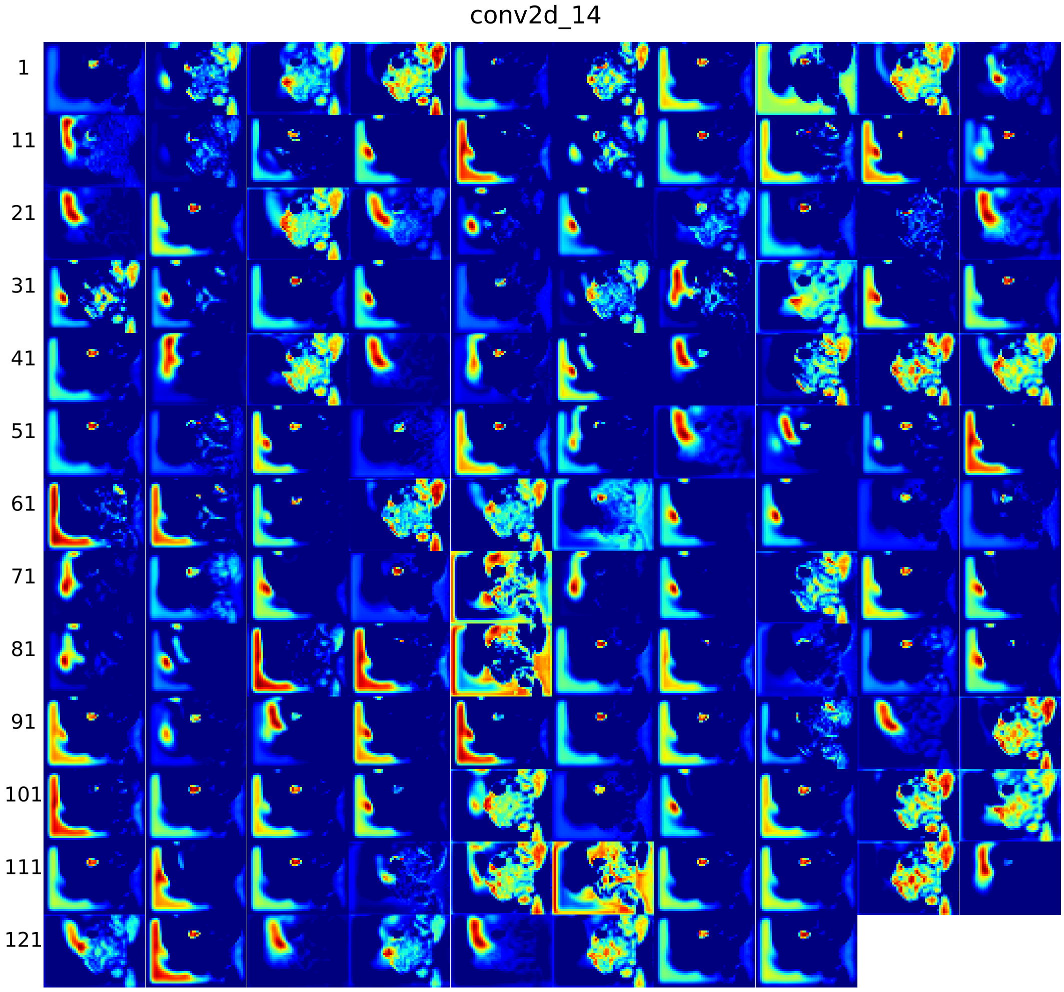
(m) Layer conv2d_12

Figure E.4: Activation maps at conv2d_12 layer (m) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



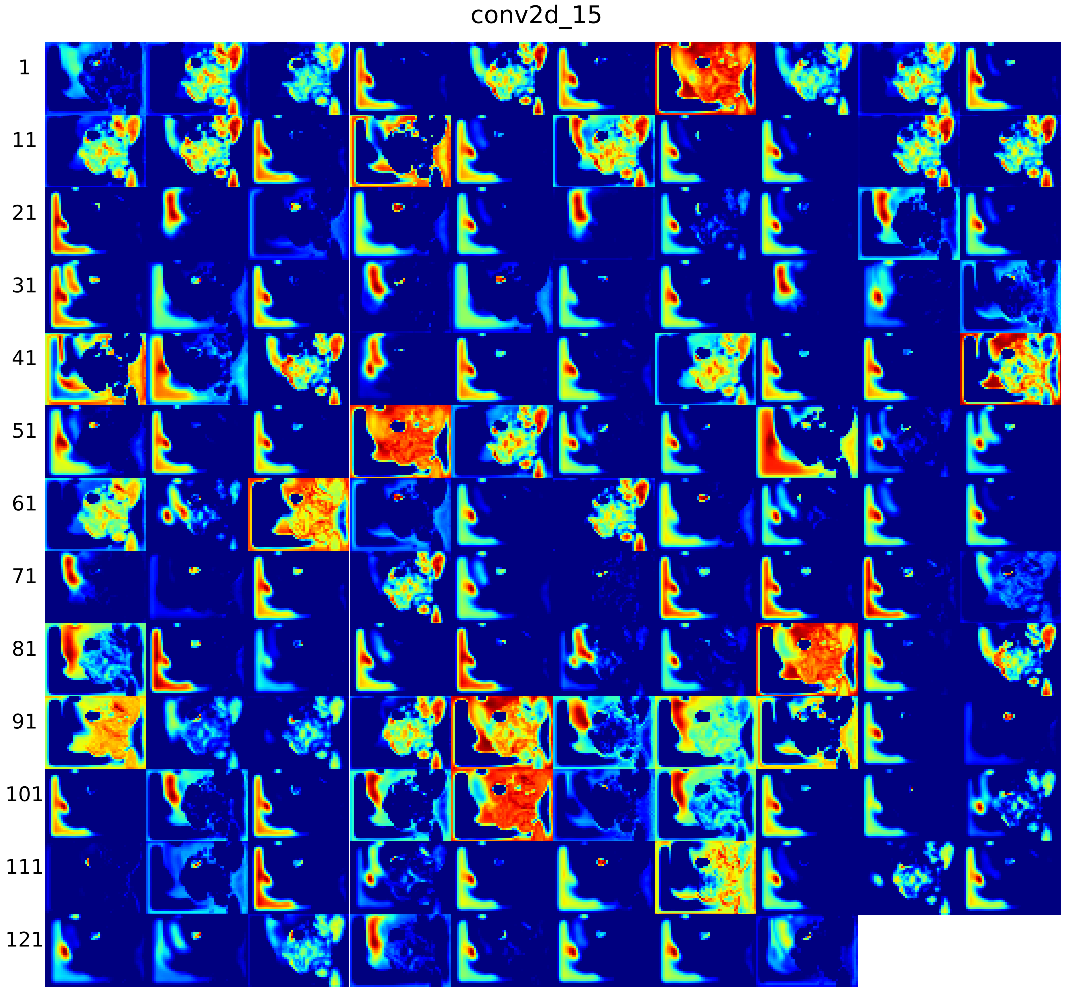
(n) Layer conv2d_13

Figure E.4: Activation maps at conv2d_13 layer (n) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



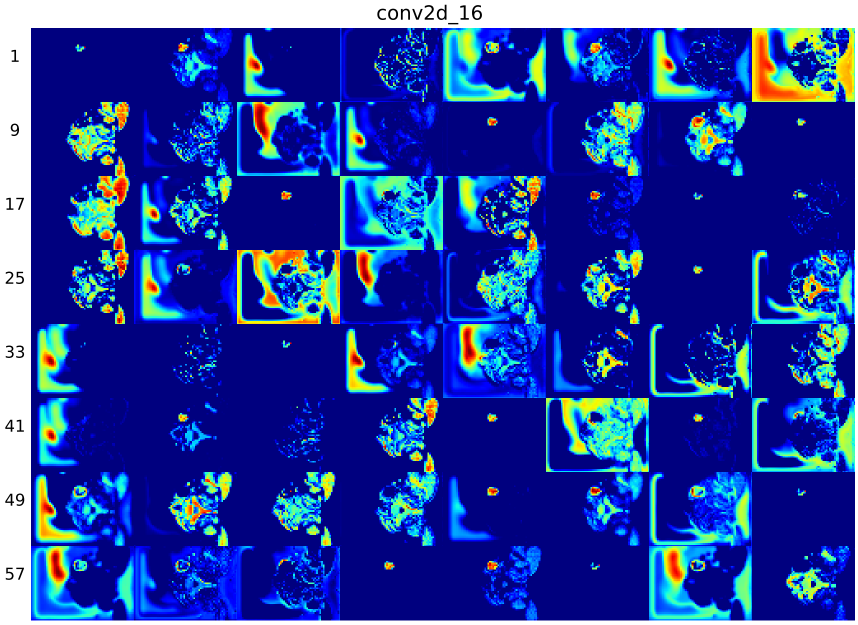
(o) Layer conv2d_14

Figure E.4: Activation maps at conv2d_14 layer (o) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.

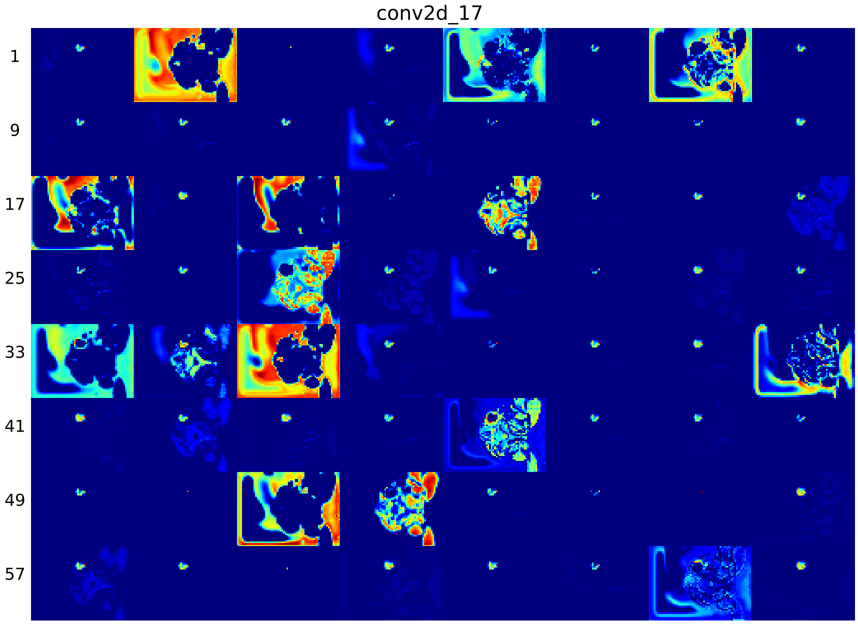


(p) Layer conv2d_15

Figure E.4: Activation maps at conv2d_15 layer (p) of image from patient 217, slice 20, with a Dice score of 0.95. Continued on next page.



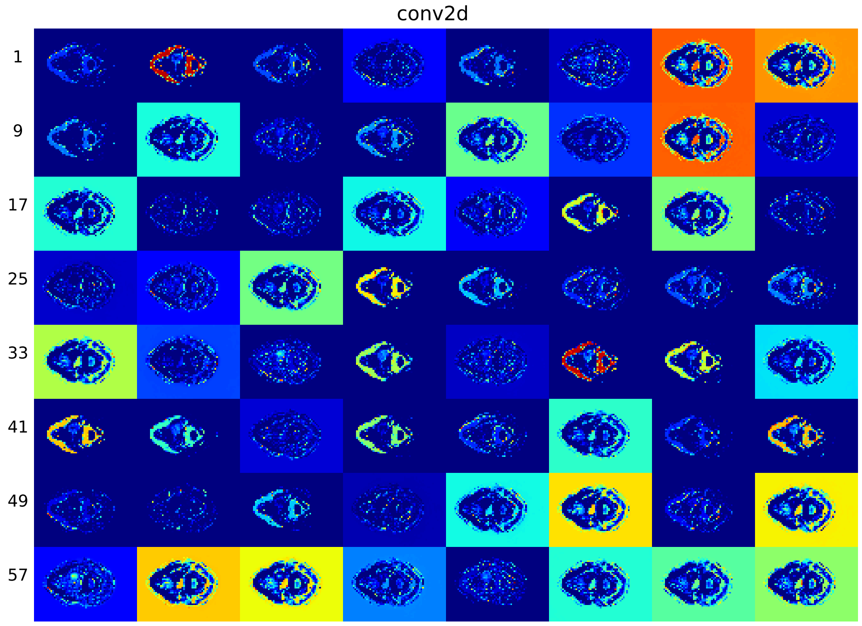
(q) Layer conv2d_16



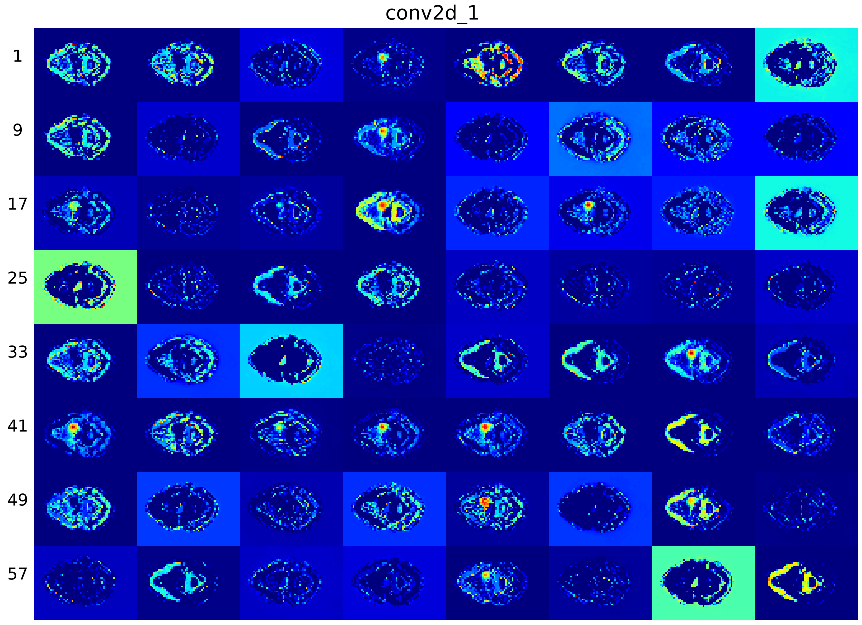
(r) Layer conv2d_17

Figure E.4: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 217, slice 20, with a Dice score of 0.95.

E.5 Activation maps patient 233, slice 48

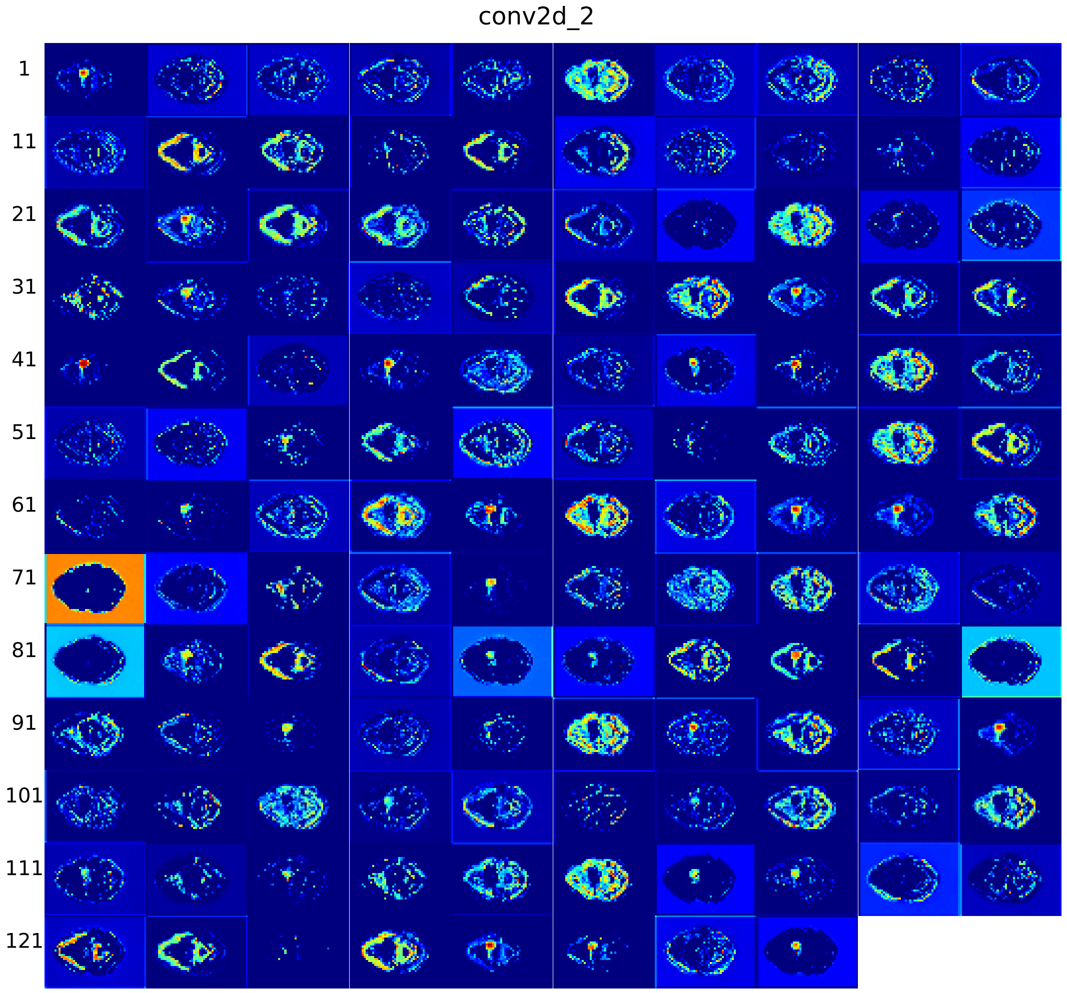


(a) Layer conv2d



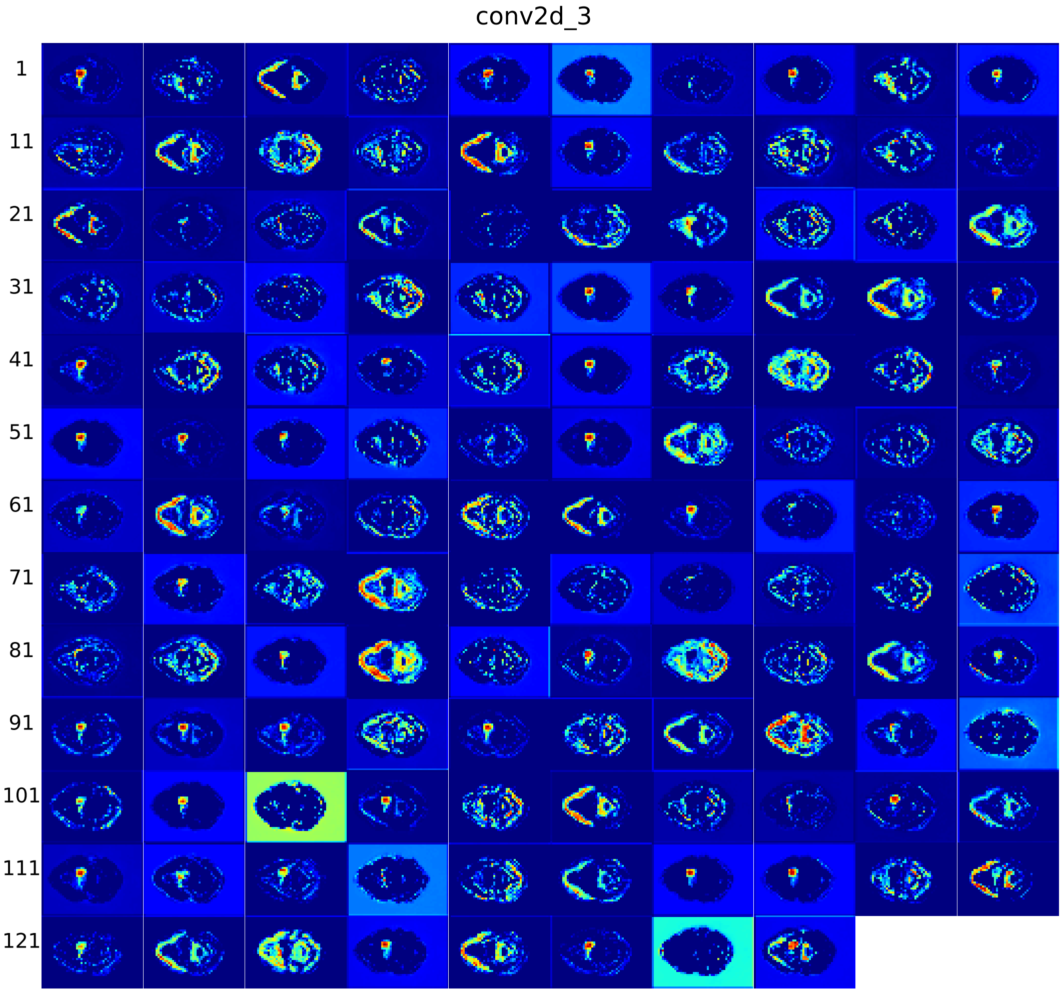
(b) Layer conv2d_1

Figure E.5: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



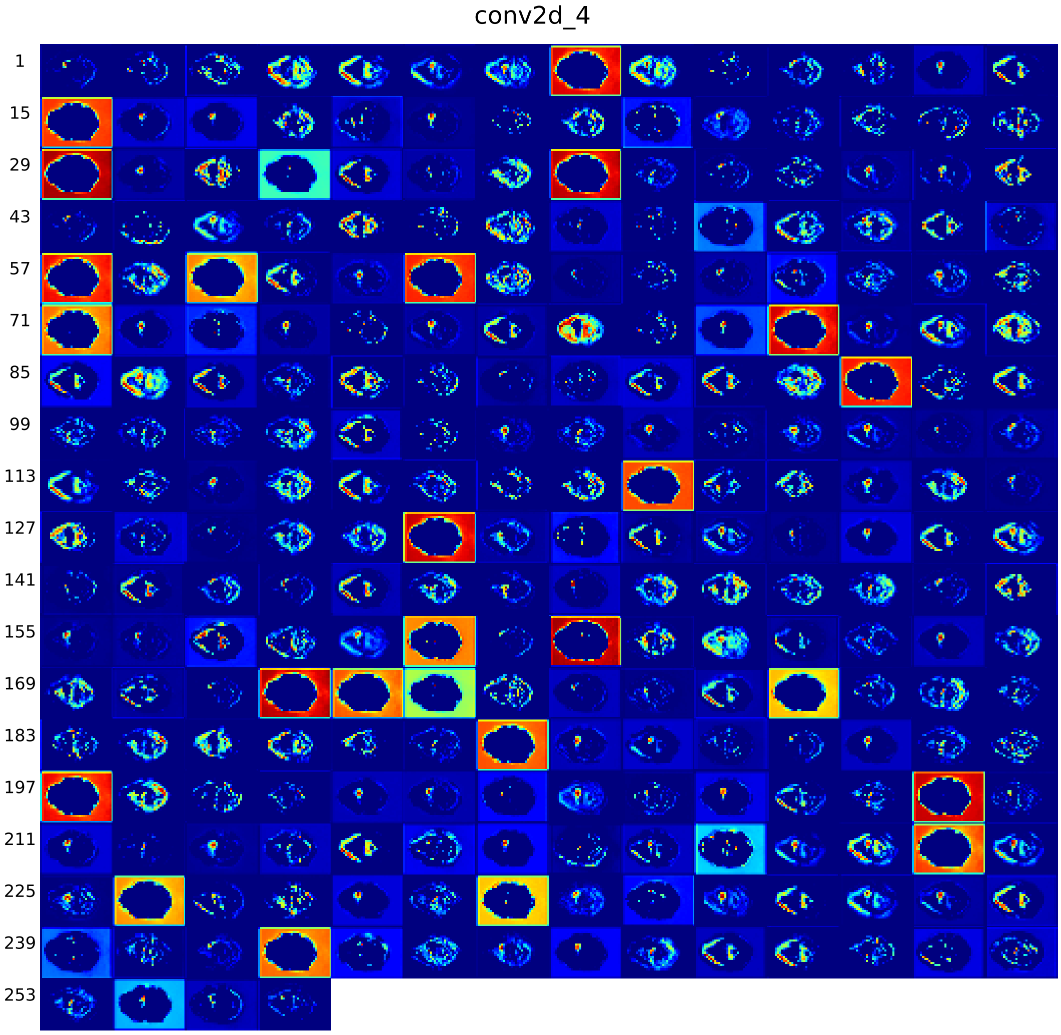
(c) Layer conv2d_2

Figure E.5: Activation maps at conv2d_2 layer (c) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



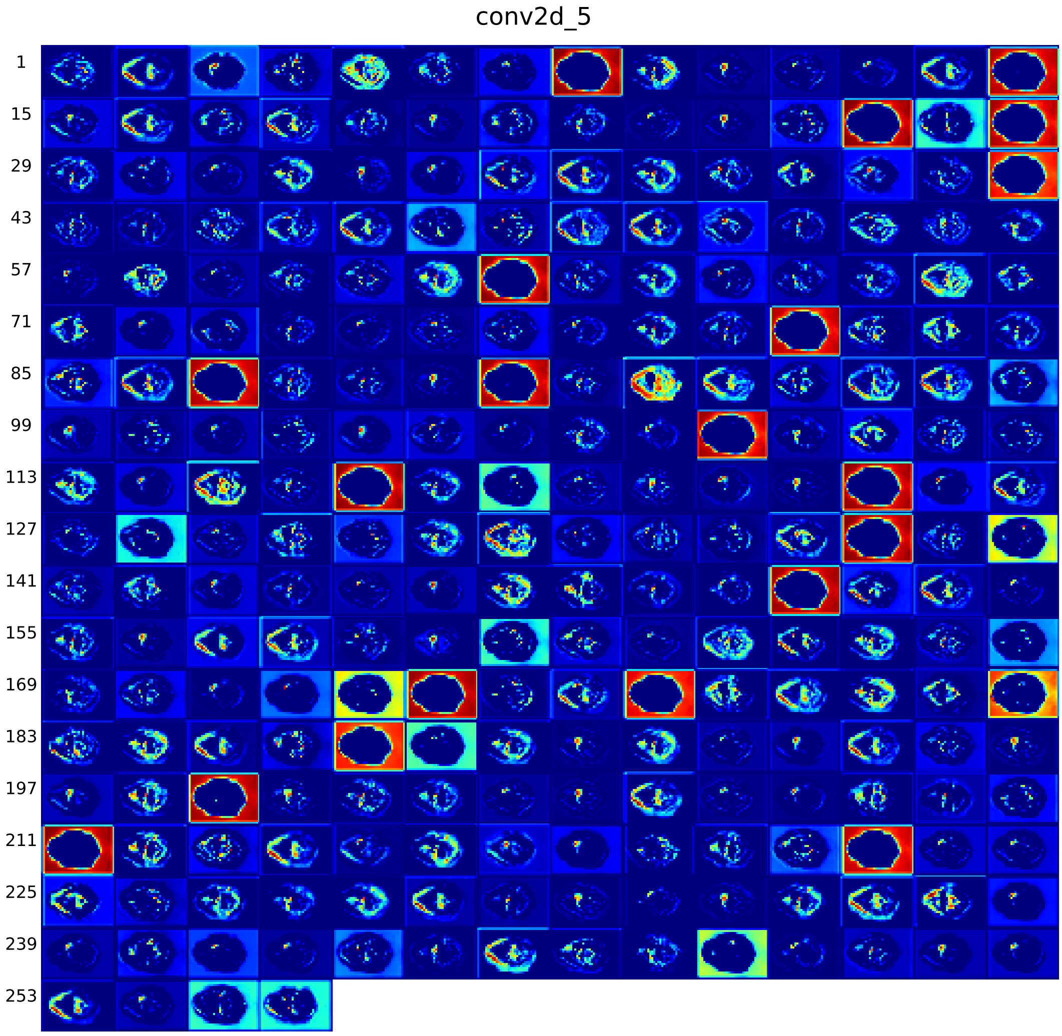
(d) Layer conv2d_3

Figure E.5: Activation maps at conv2d_3 layer (d) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



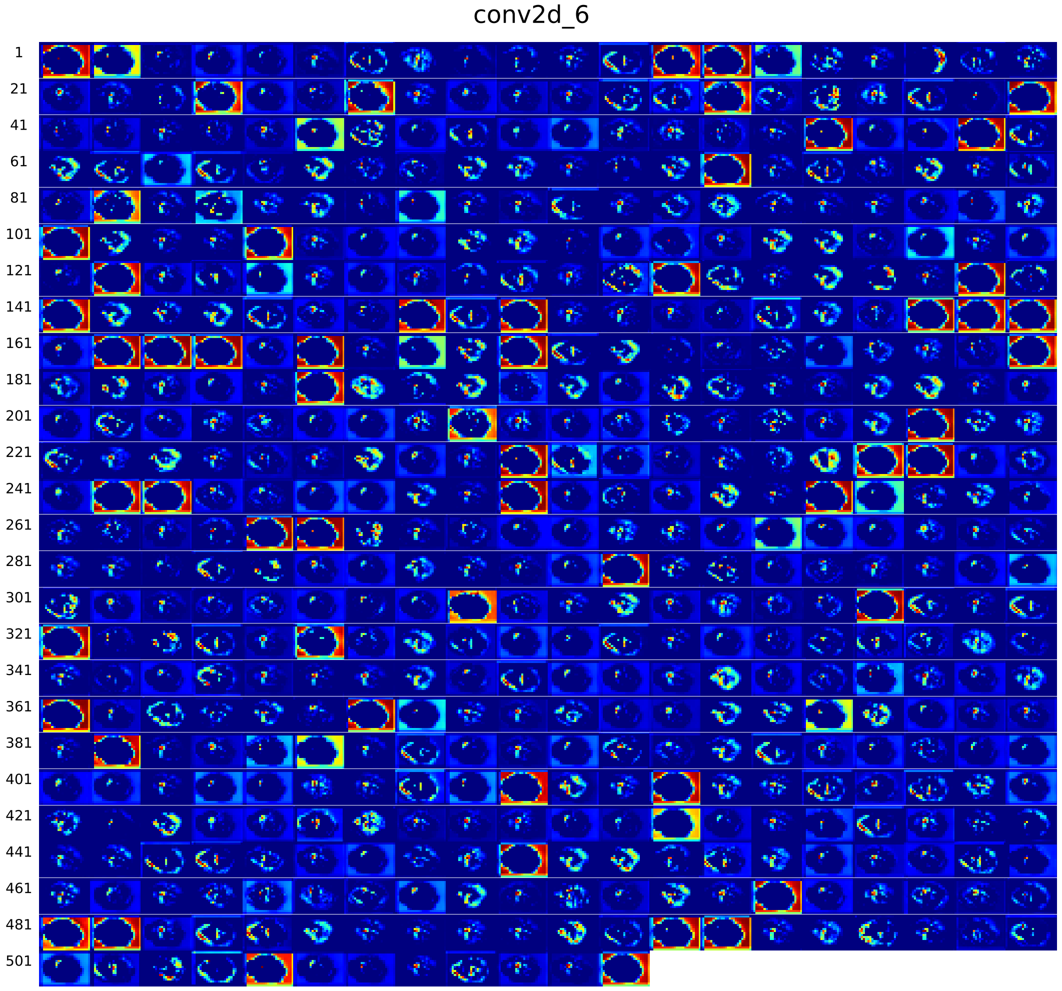
(e) Layer conv2d_4

Figure E.5: Activation maps at conv2d_4 layer (e) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



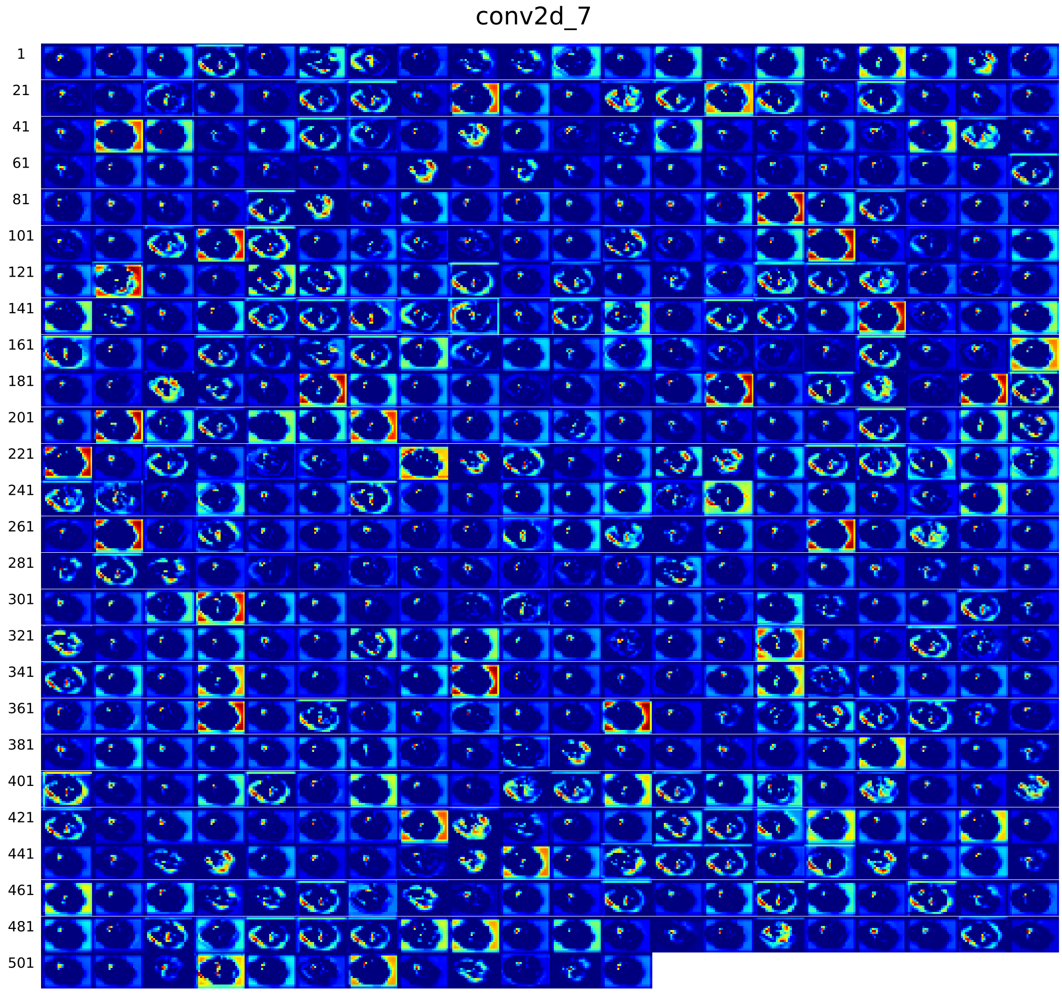
(f) Layer conv2d_5

Figure E.5: Activation maps at conv2d_5 layer (f) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



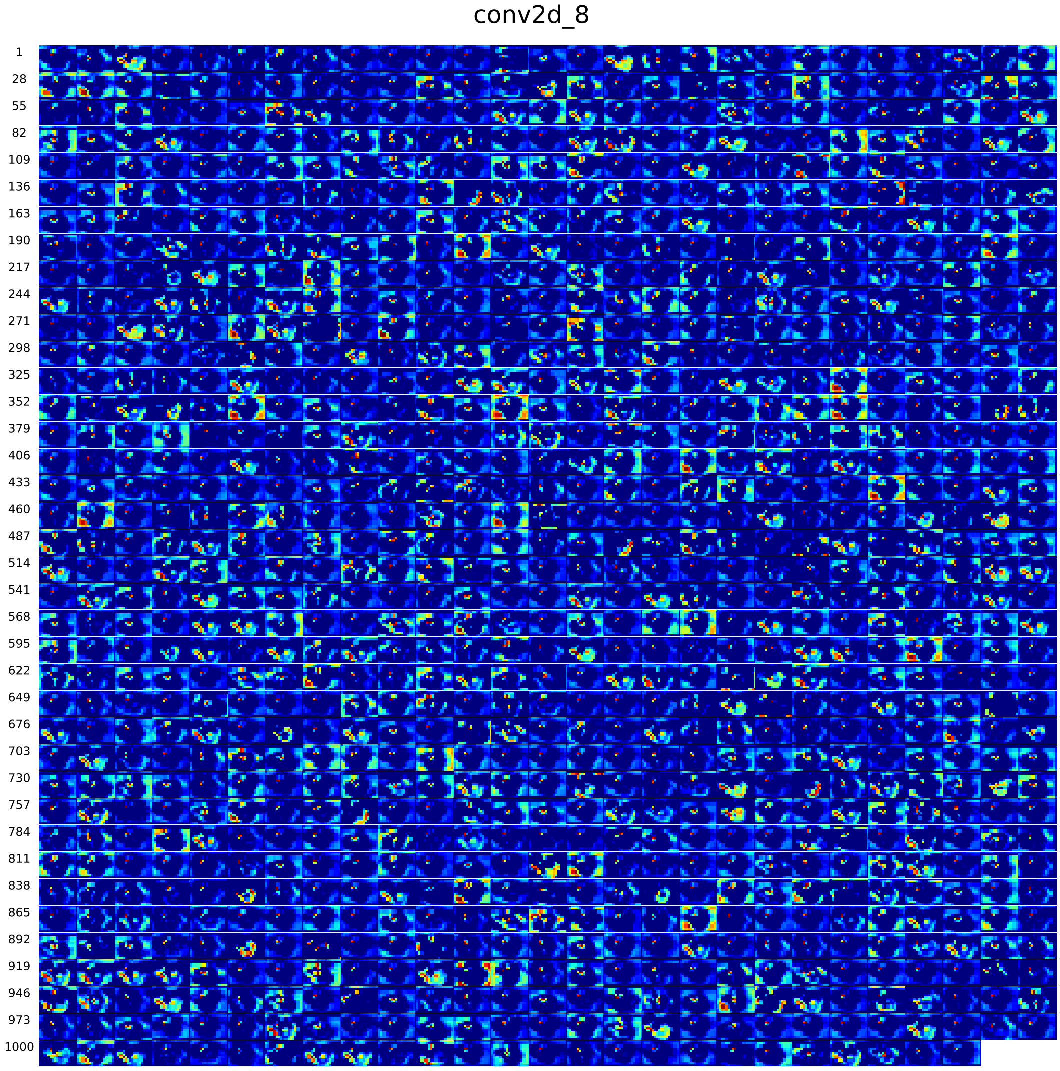
(g) Layer conv2d_6

Figure E.5: Activation maps at conv2d_6 layer (g) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



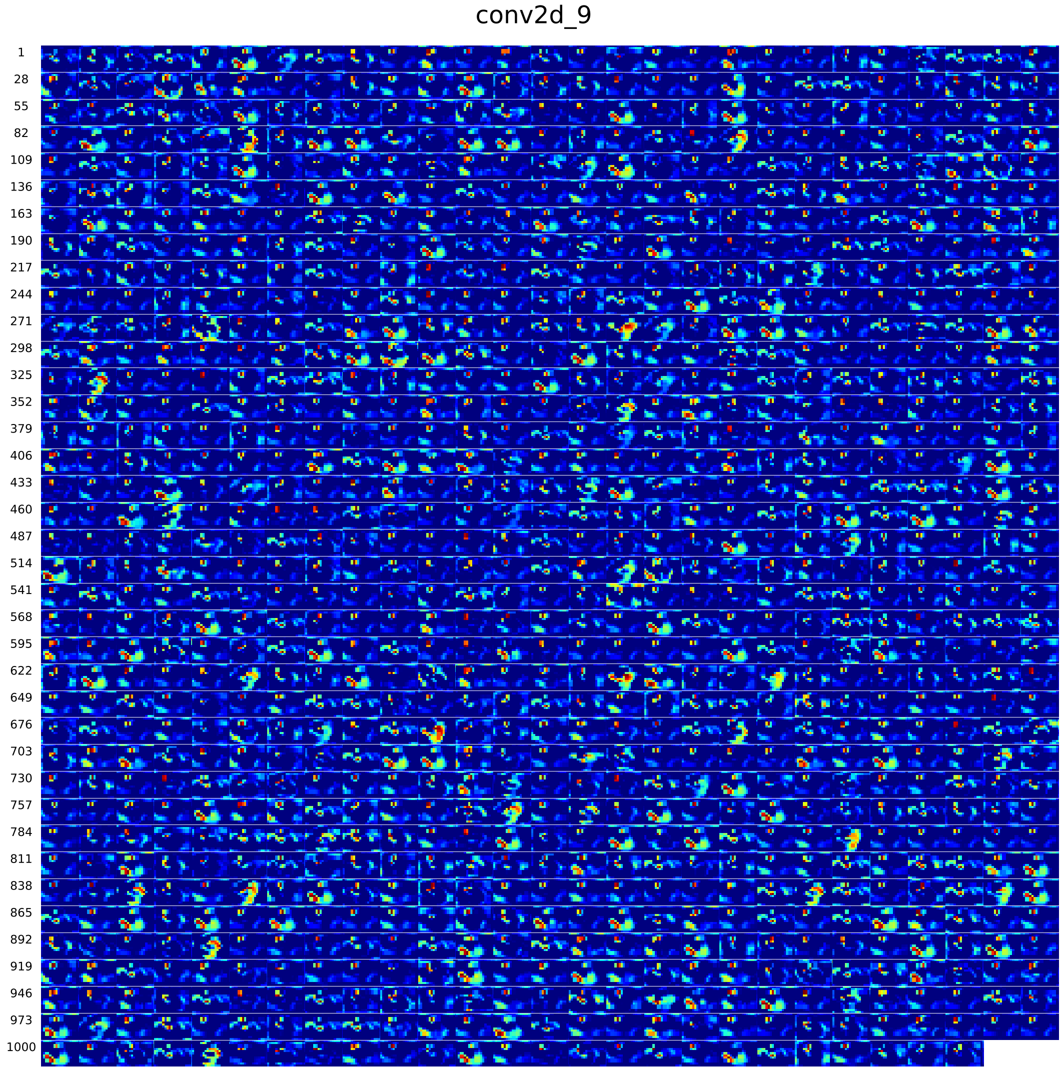
(h) Layer conv2d_7

Figure E.5: Activation maps at conv2d_7 layer (h) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



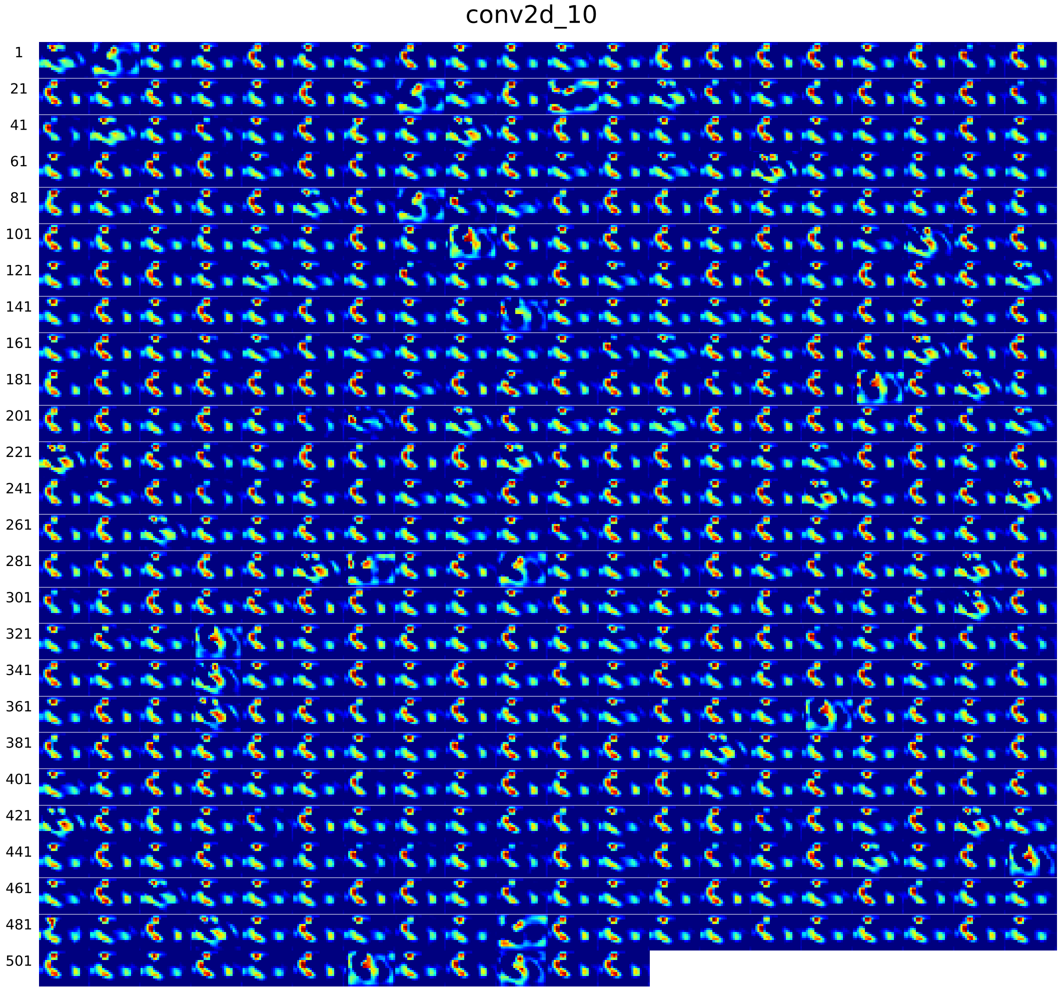
(i) Layer conv2d_8

Figure E.5: Activation maps at conv2d_8 layer (i) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



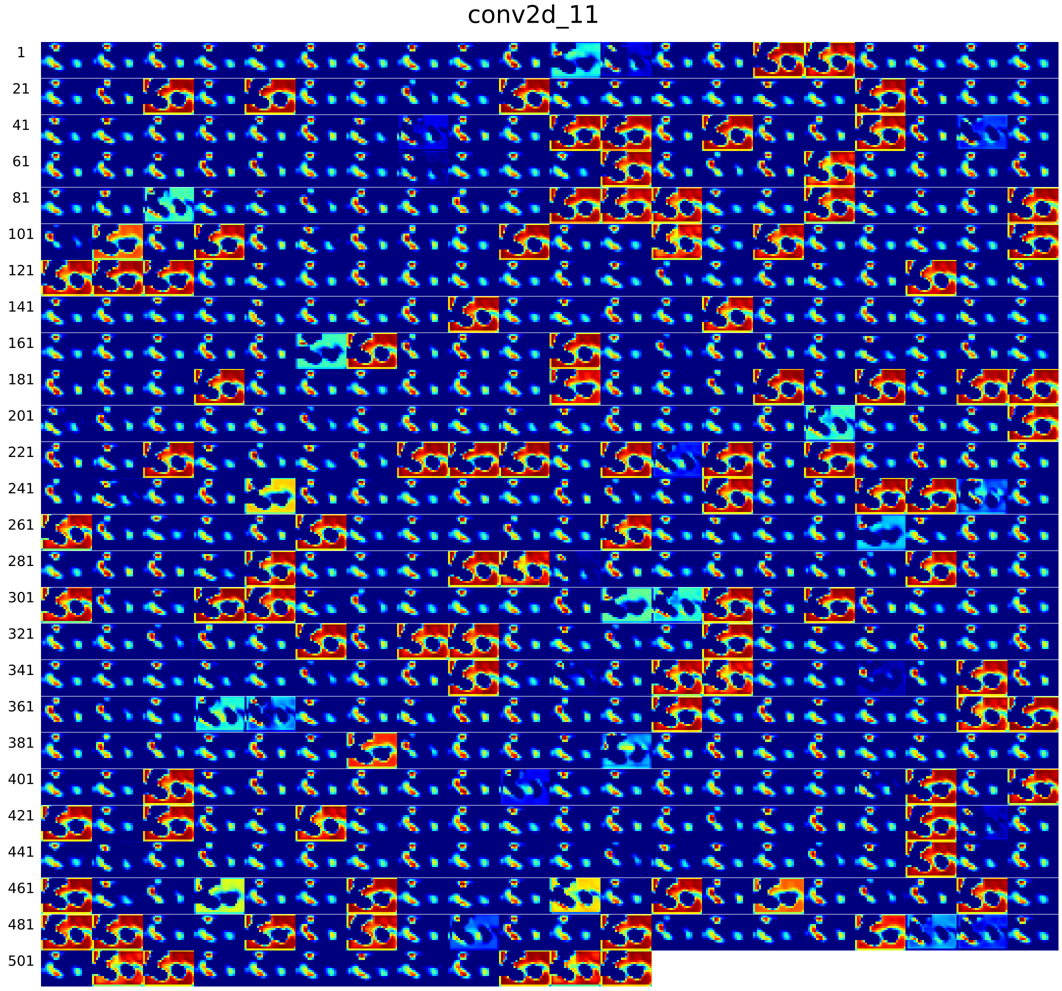
(j) Layer conv2d_9

Figure E.5: Activation maps at conv2d_9 layer (j) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



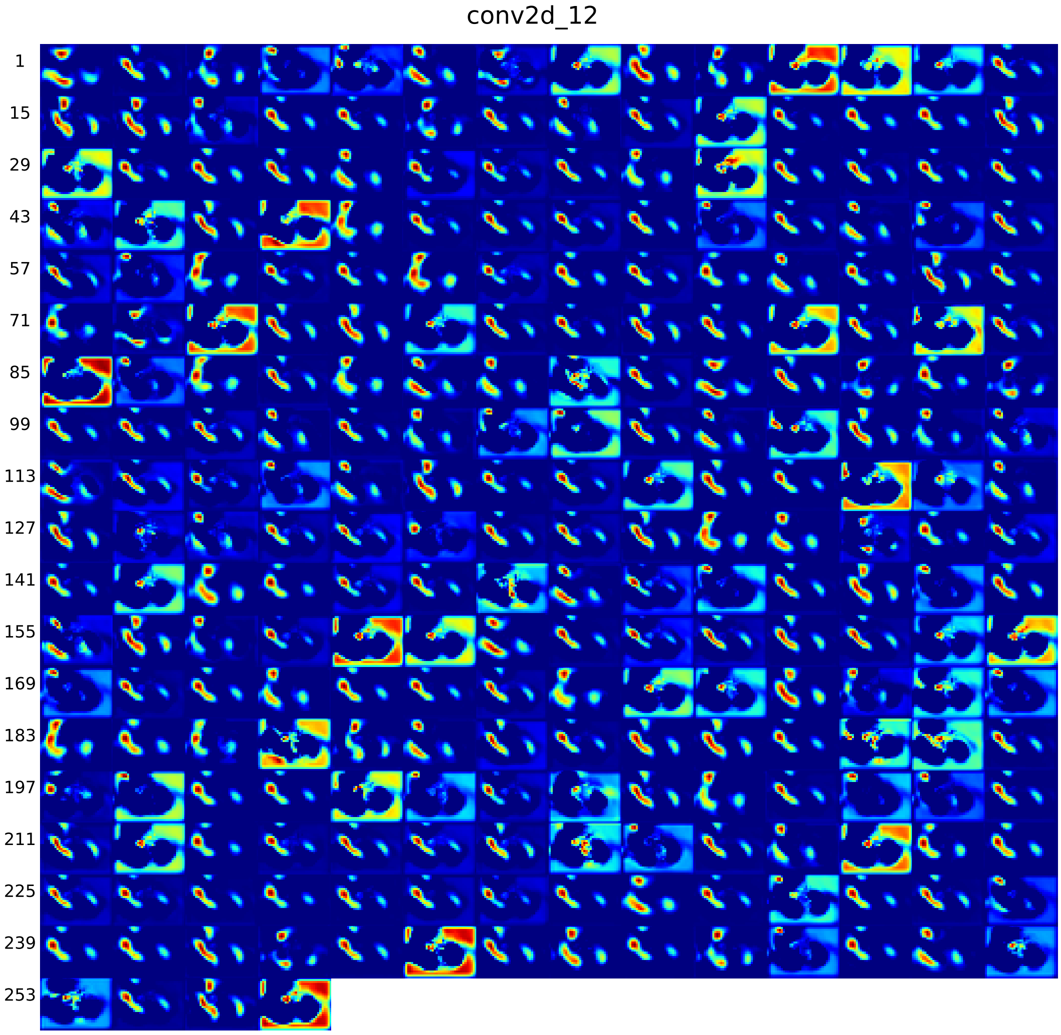
(k) Layer conv2d_10

Figure E.5: Activation maps at conv2d_10 layer (k) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



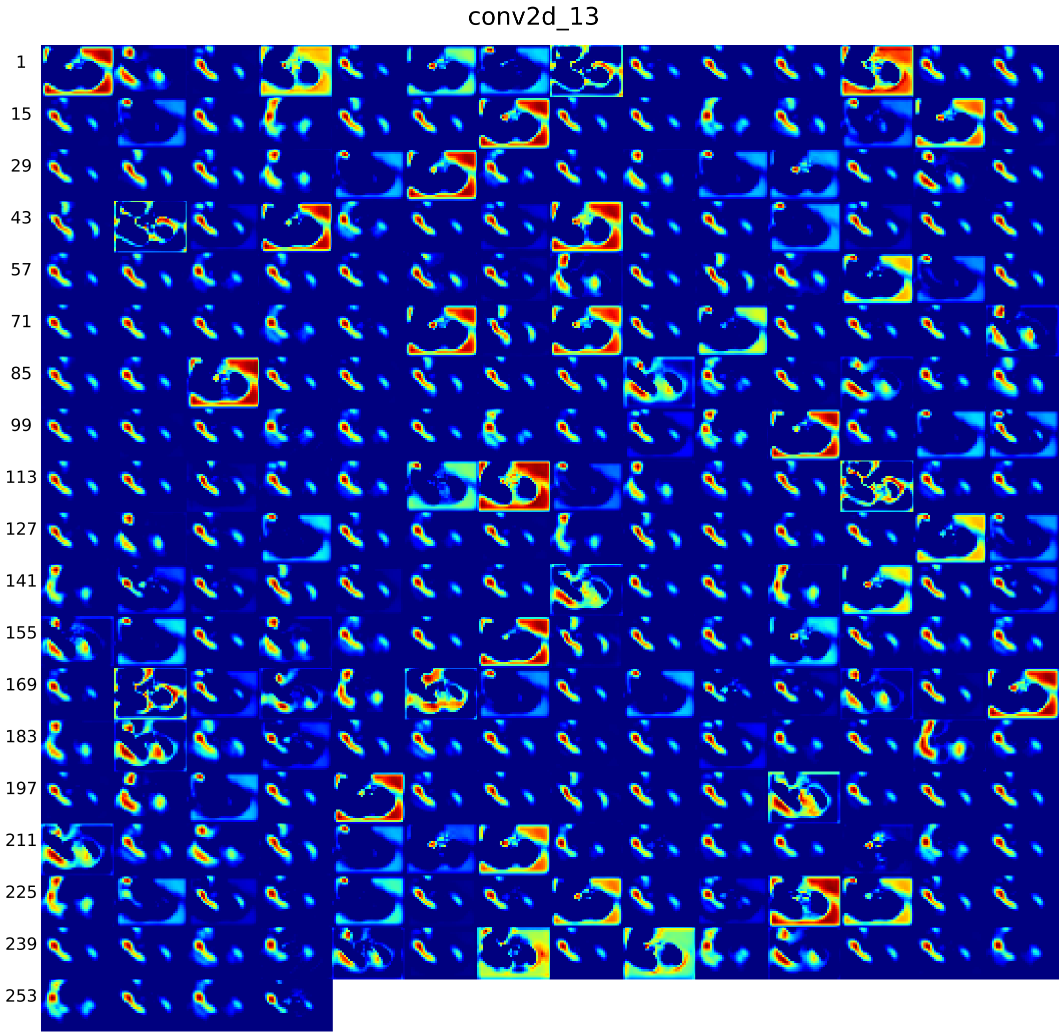
(l) Layer conv2d_11

Figure E.5: Activation maps at conv2d_11 layer (l) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page..



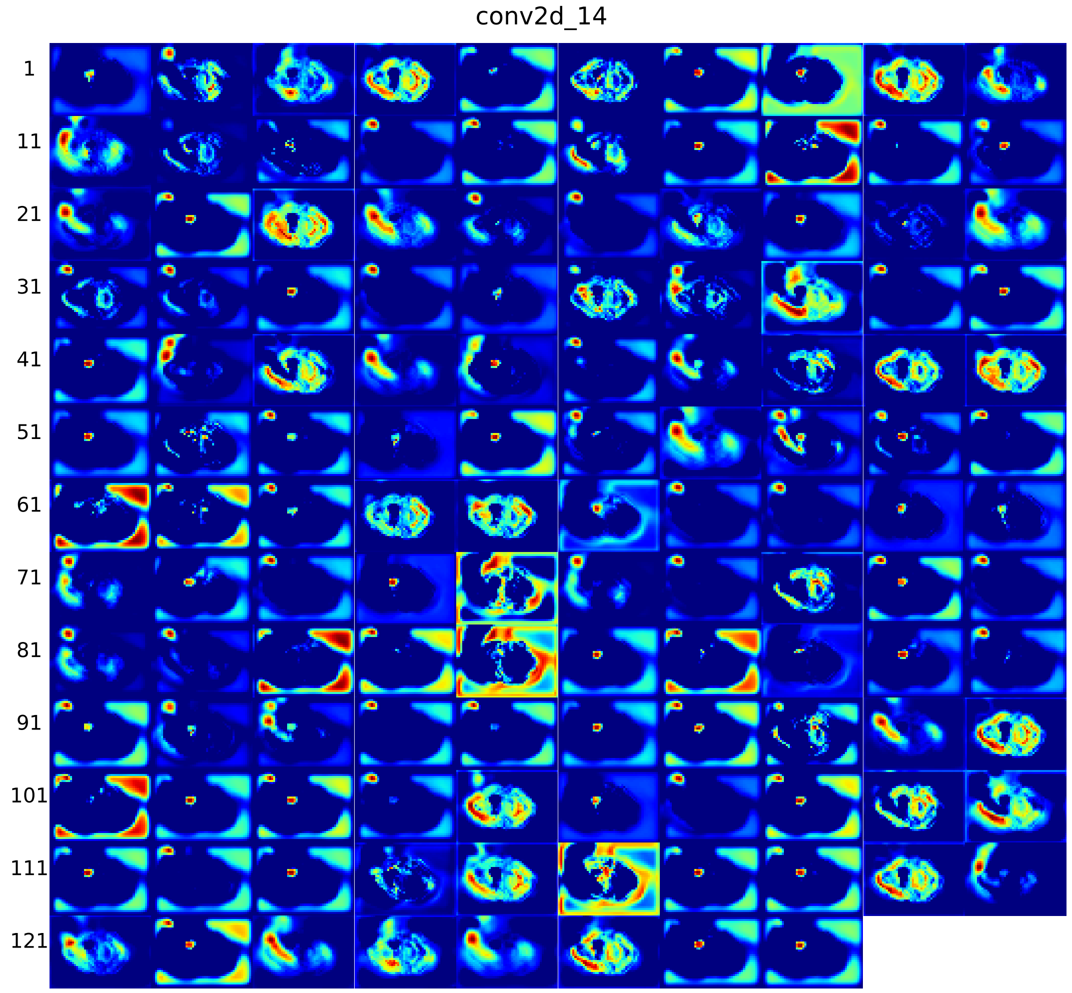
(m) Layer conv2d_12

Figure E.5: Activation maps at conv2d_12 layer (m) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



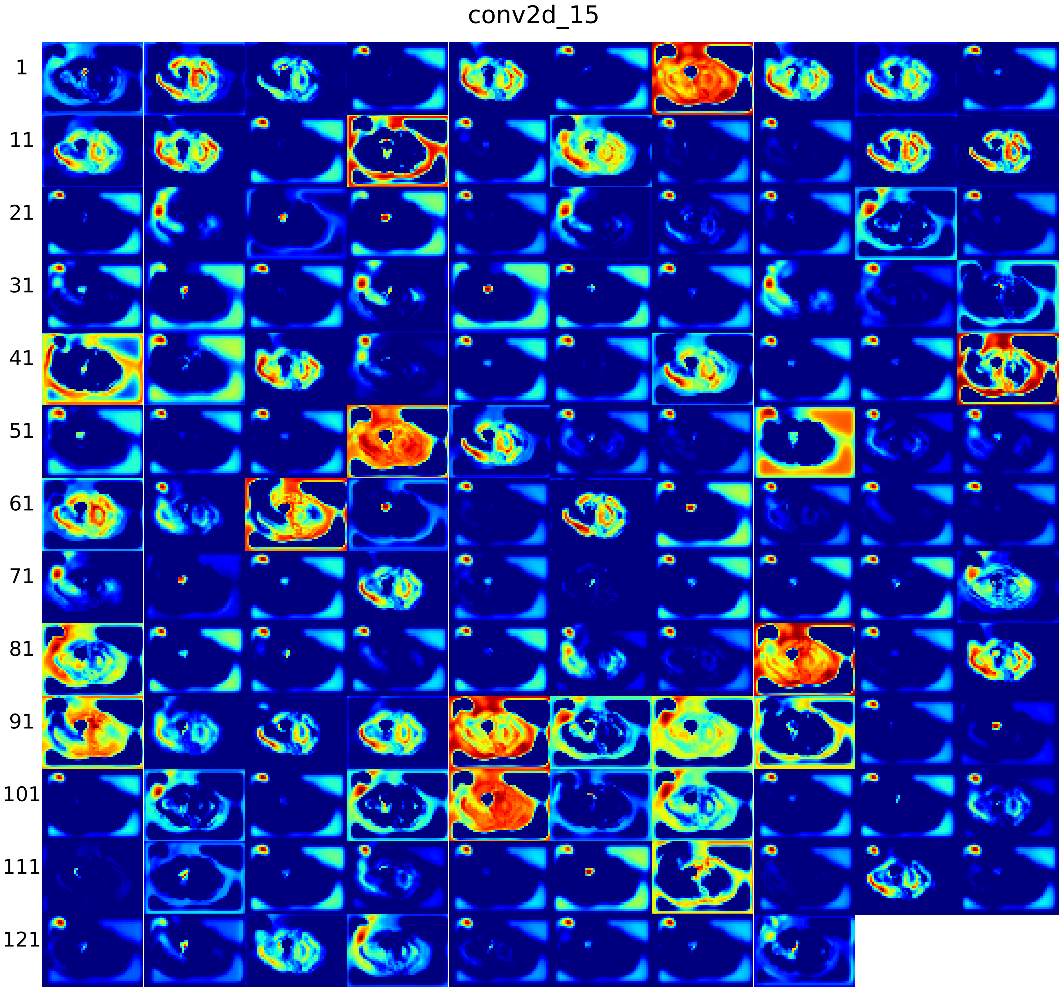
(n) Layer conv2d_13

Figure E.5: Activation maps at conv2d_13 layer (n) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



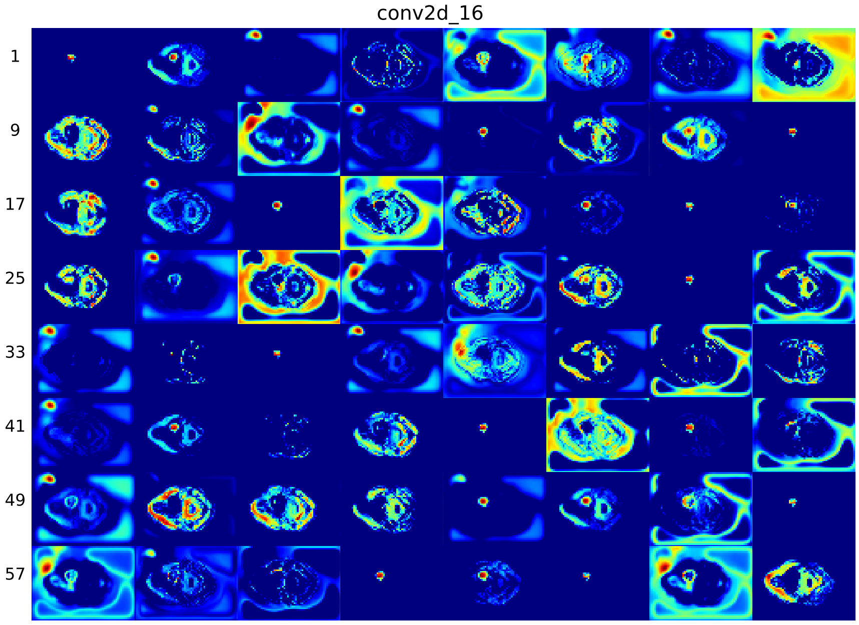
(o) Layer conv2d_14

Figure E.5: Activation maps at conv2d_14 layer (o) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.

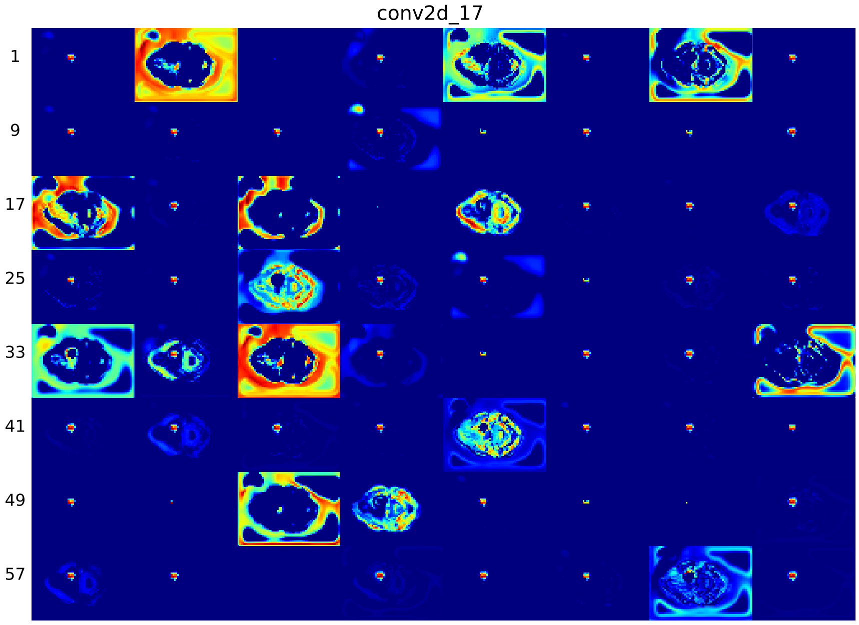


(p) Layer conv2d_15

Figure E.5: Activation maps at conv2d_15 layer (p) of image from patient 233, slice 48, with a Dice score of 0.95. Continued on next page.



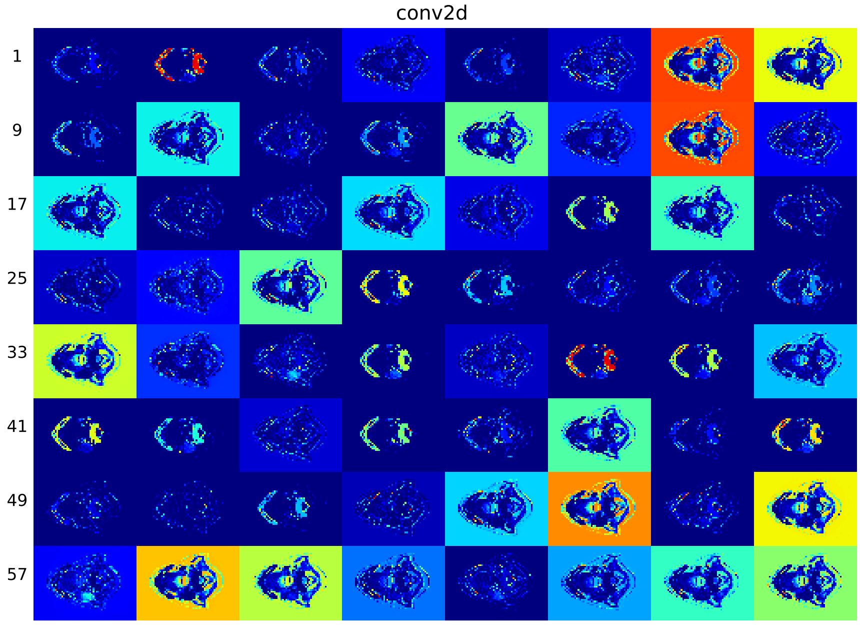
(q) Layer conv2d_16



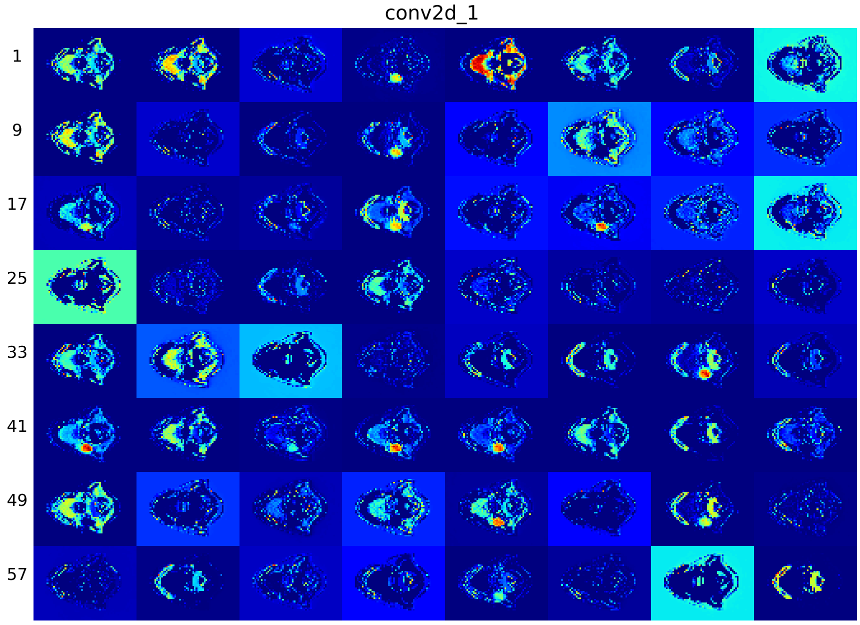
(r) Layer conv2d_17

Figure E.5: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 233, slice 48, with a Dice score of 0.95.

E.6 Activation maps patient 249, slice 55

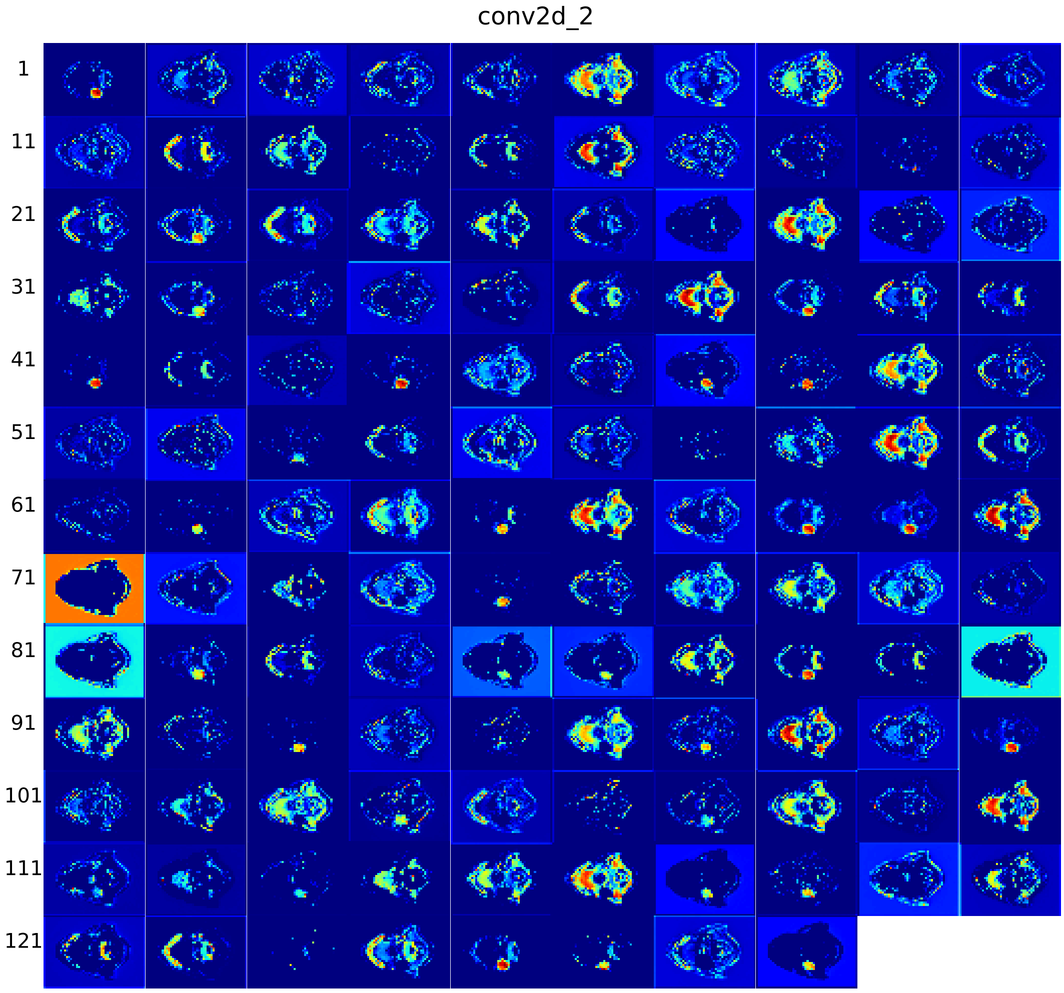


(a) Layer conv2d



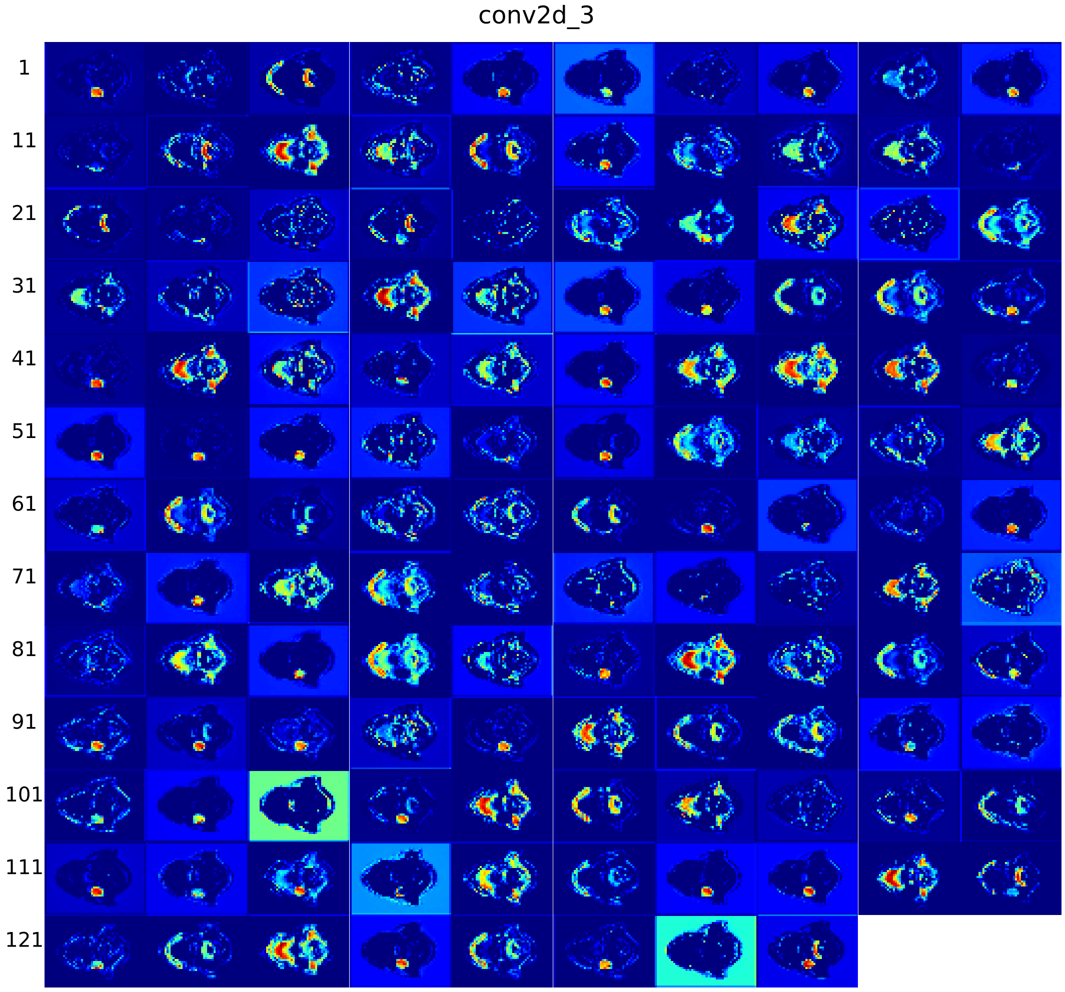
(b) Layer conv2d_1

Figure E.6: Activation maps at conv2d layer (a) and conv2d_1 layer (b) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



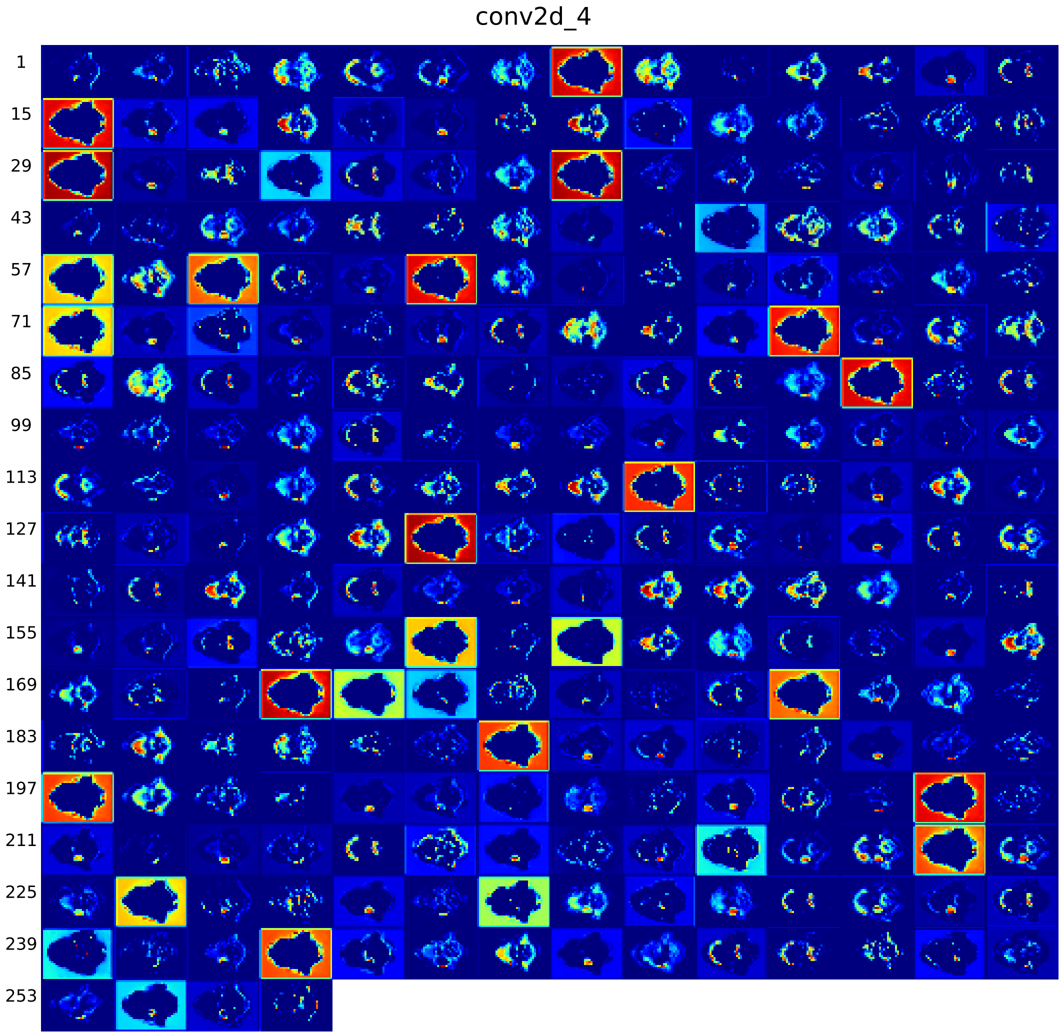
(c) Layer conv2d_2

Figure E.6: Activation maps at conv2d_2 layer (c) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



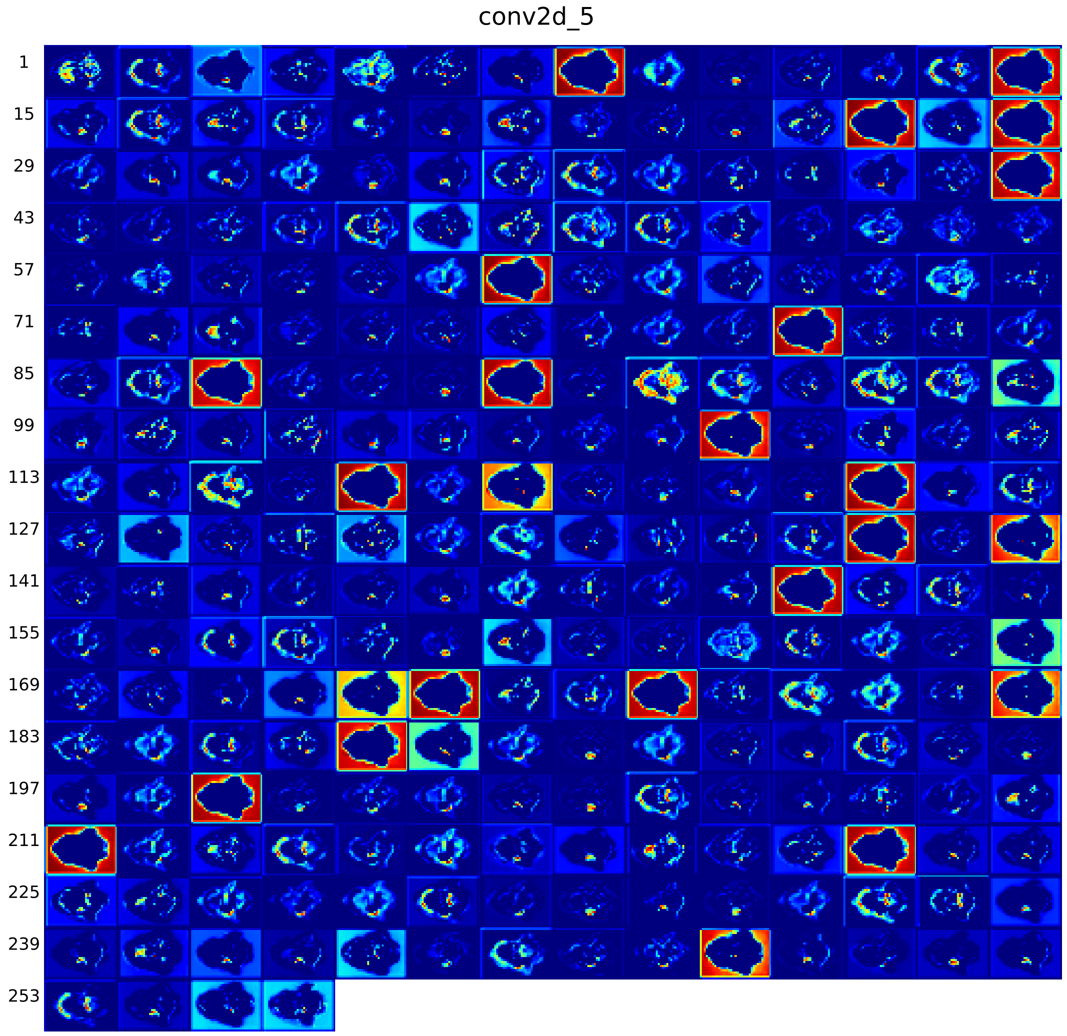
(d) Layer conv2d_3

Figure E.6: Activation maps at conv2d_3 layer (d) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



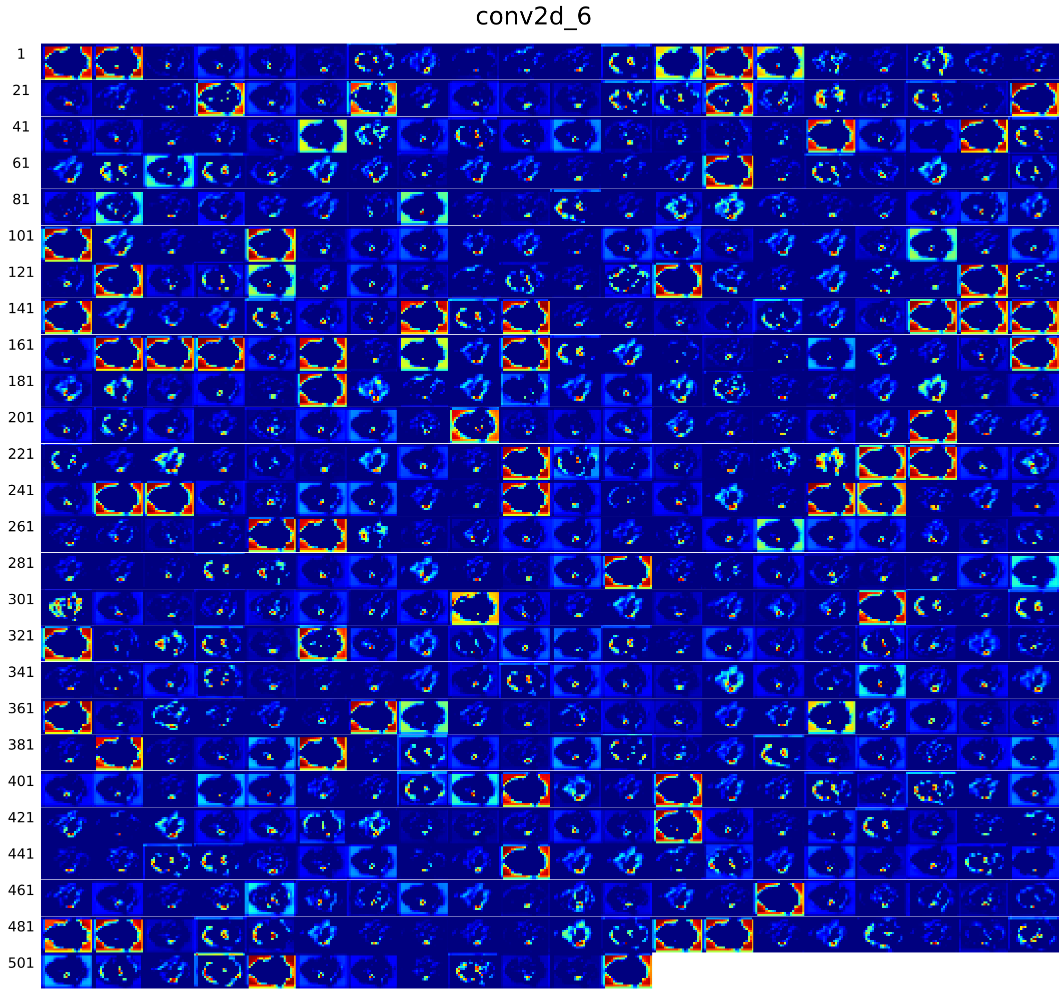
(e) Layer conv2d_4

Figure E.6: Activation maps at conv2d_4 layer (e) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



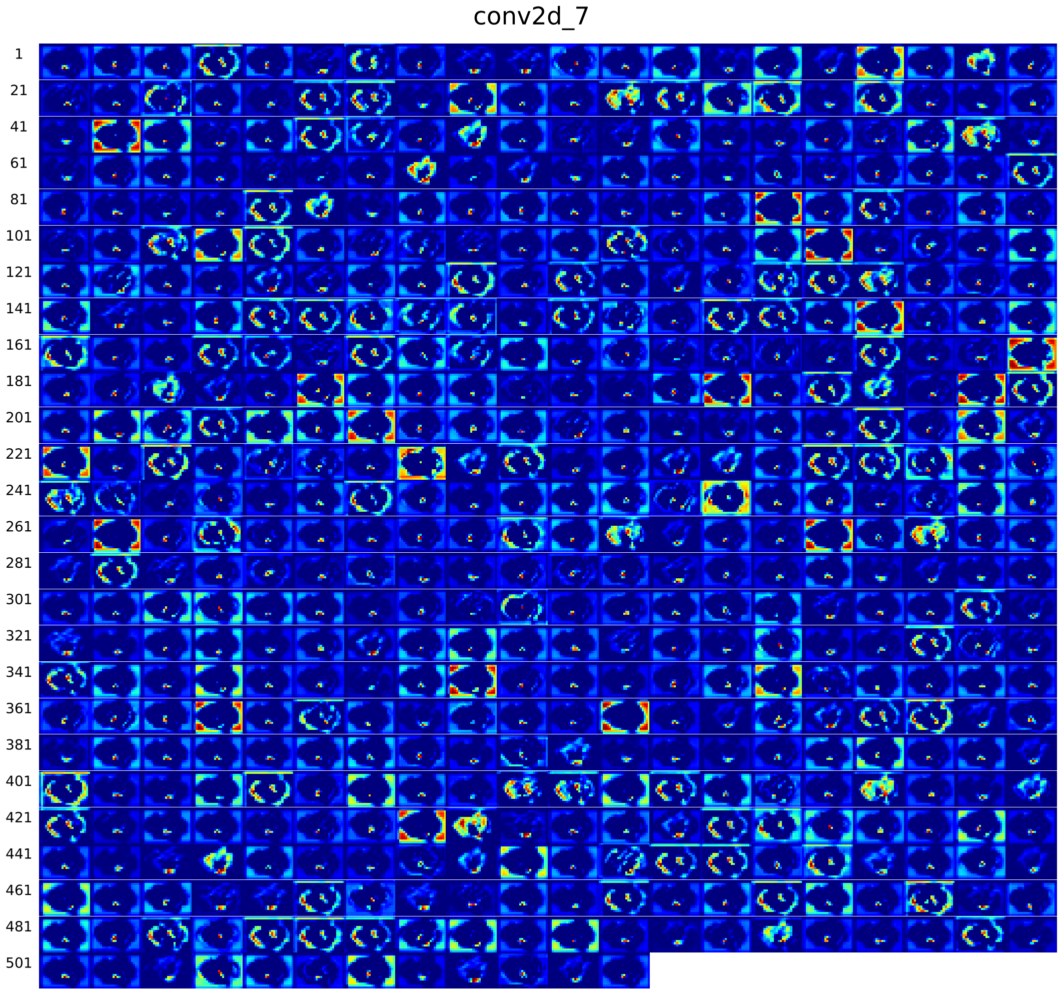
(f) Layer conv2d_5

Figure E.6: Activation maps at conv2d_5 layer (f) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



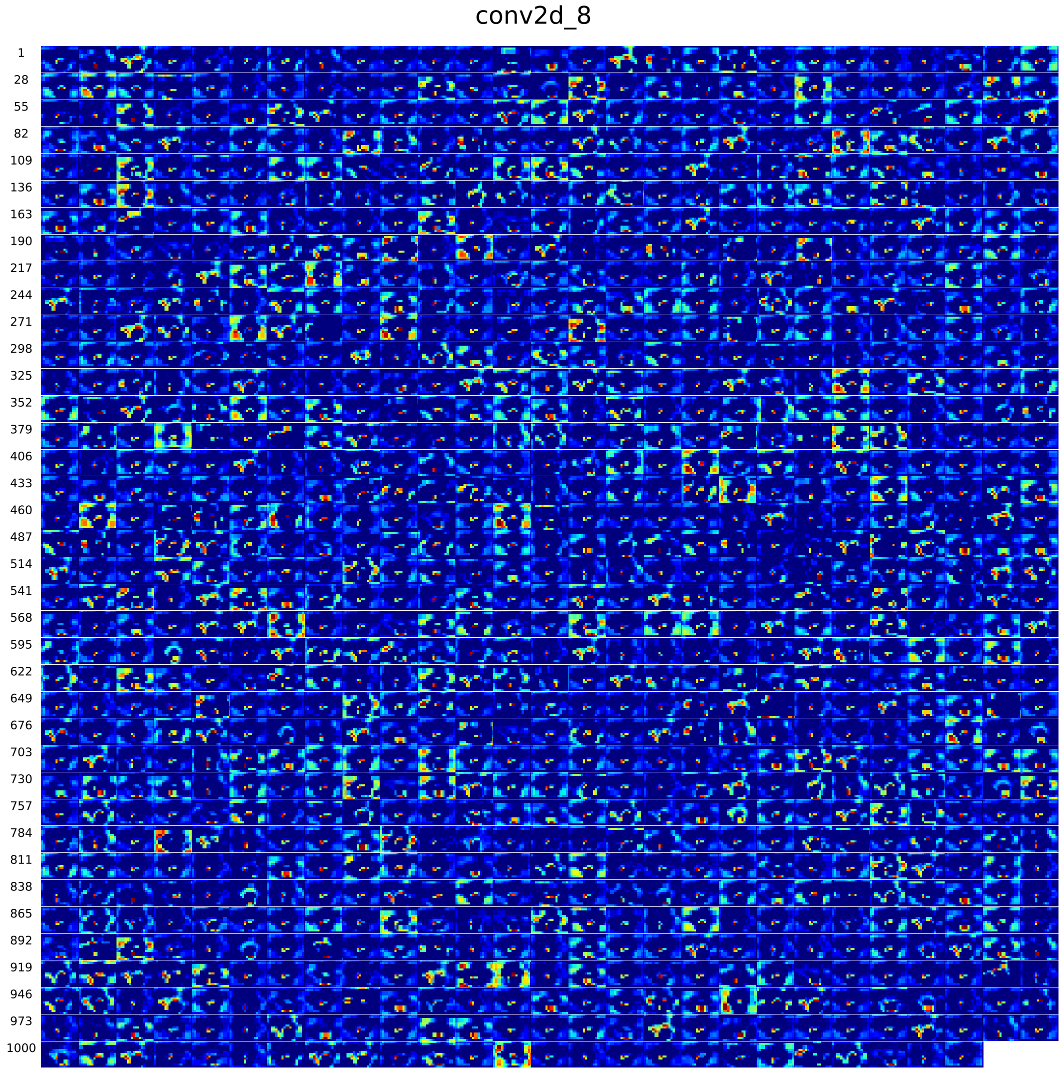
(g) Layer conv2d_6

Figure E.6: Activation maps at conv2d_6 layer (g) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



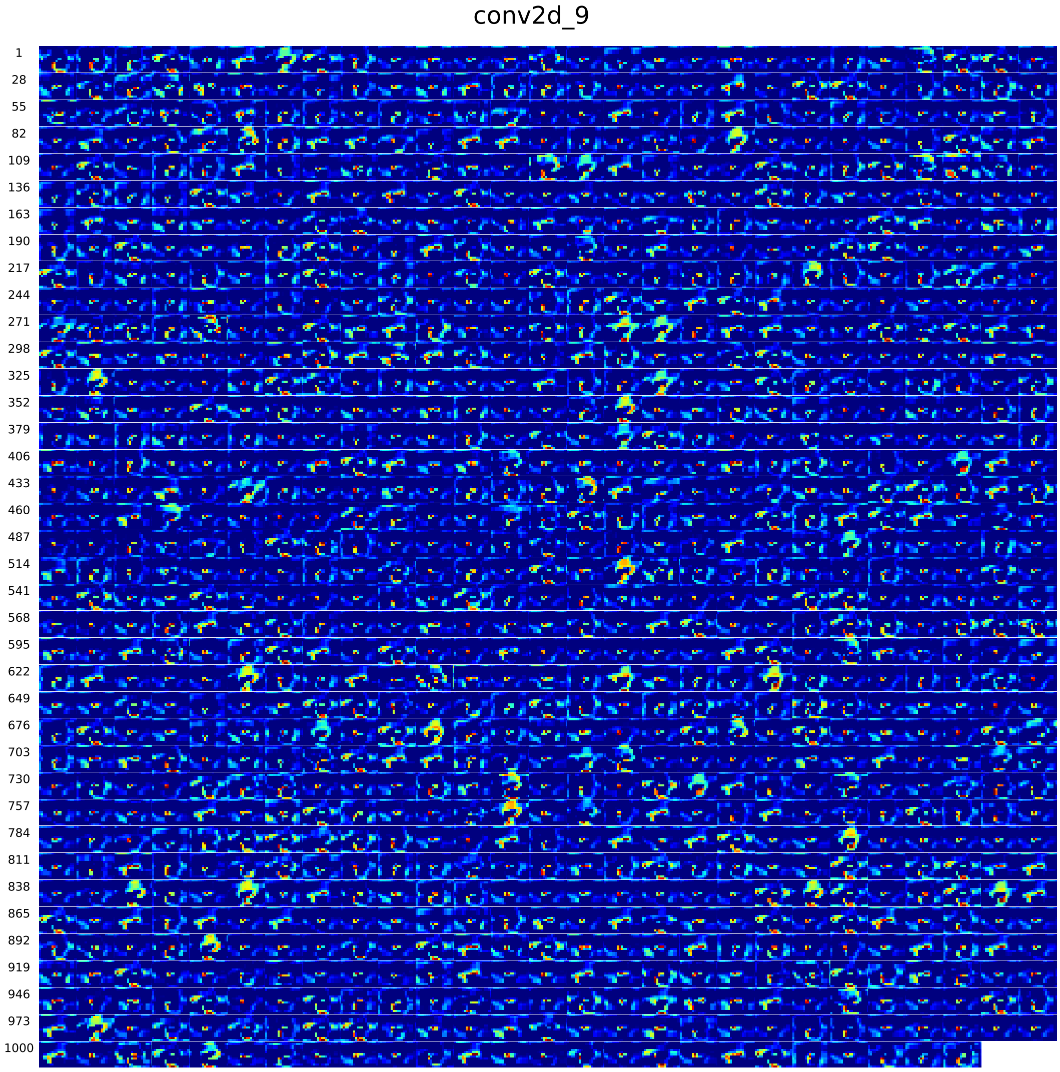
(h) Layer conv2d_7

Figure E.6: Activation maps at conv2d_7 layer (h) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



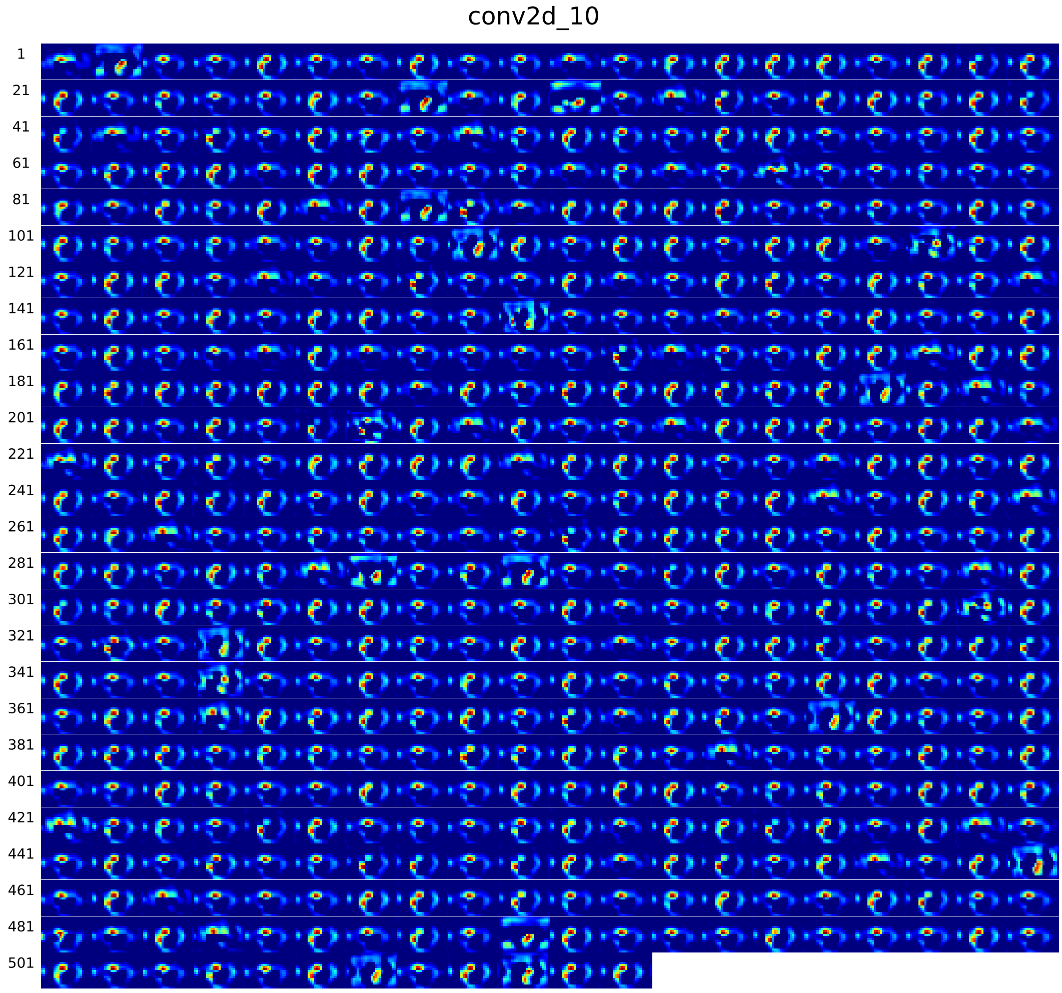
(i) Layer conv2d_8

Figure E.6: Activation maps at conv2d_8 layer (i) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



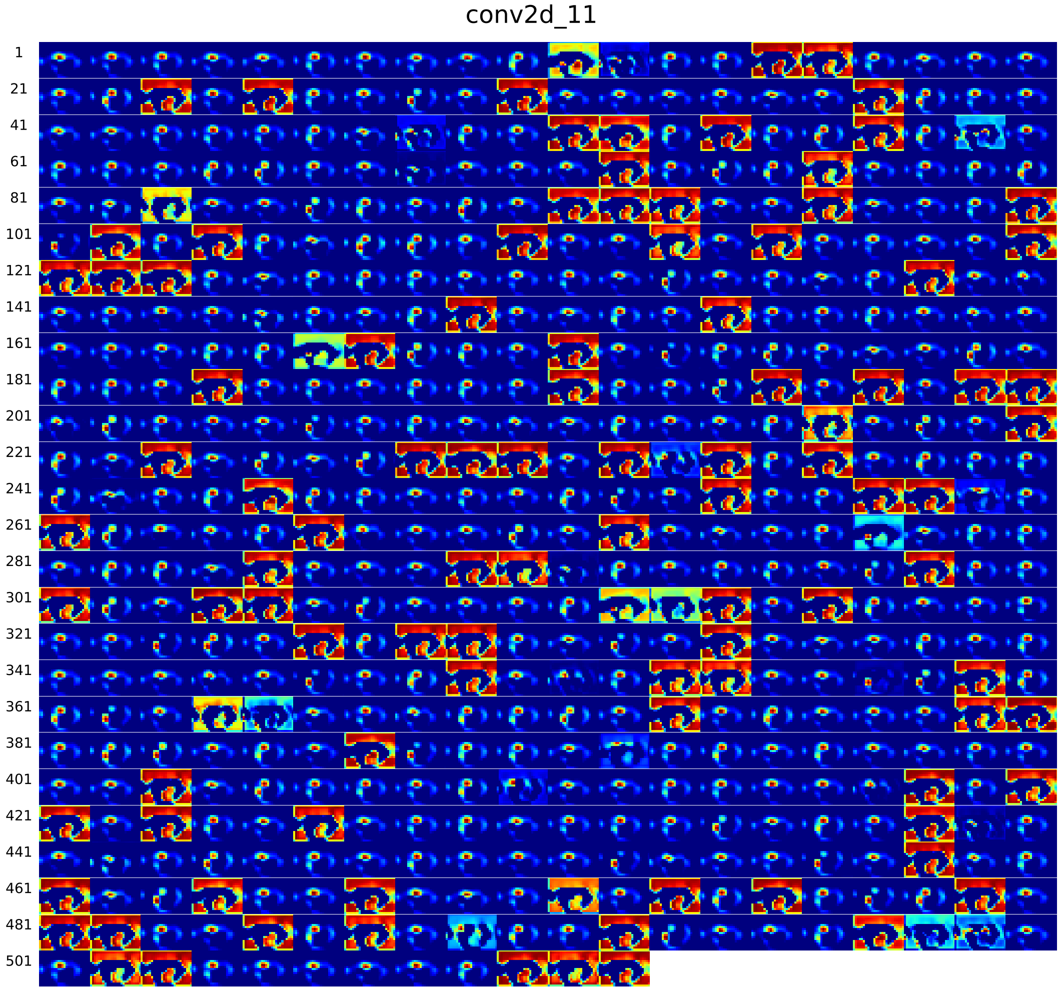
(j) Layer conv2d_9

Figure E.6: Activation maps at conv2d_9 layer (j) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



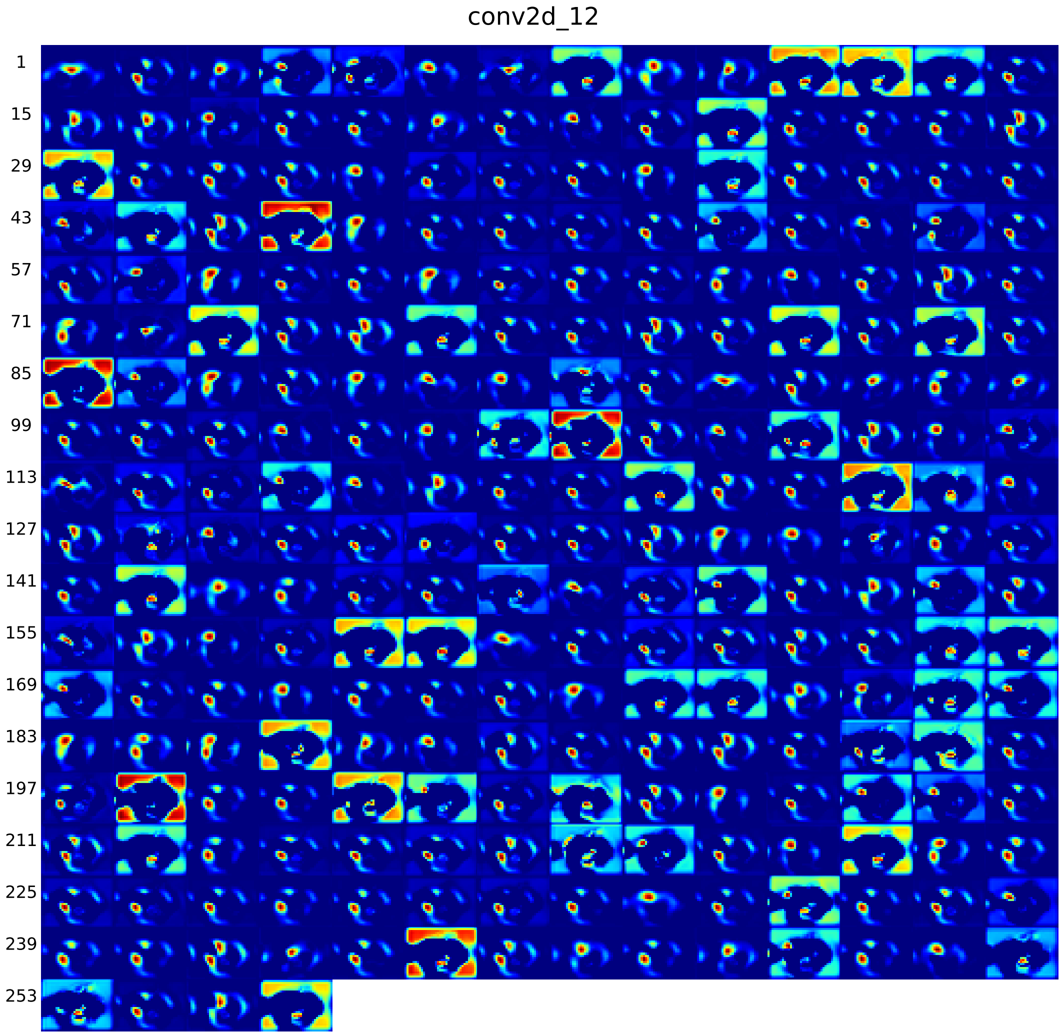
(k) Layer conv2d_10

Figure E.6: Activation maps at conv2d_10 layer (k) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



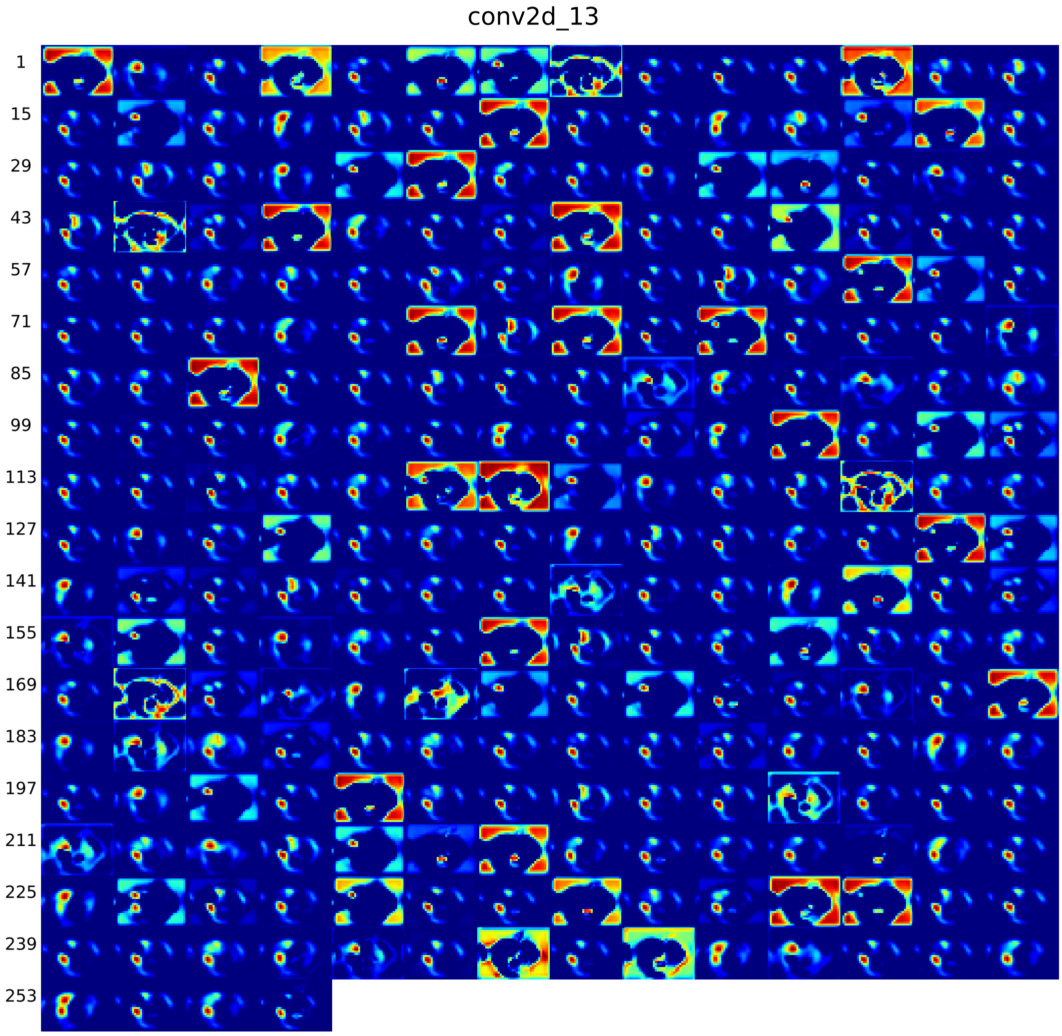
(l) Layer conv2d_11

Figure E.6: Activation maps at conv2d_11 layer (l) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page..



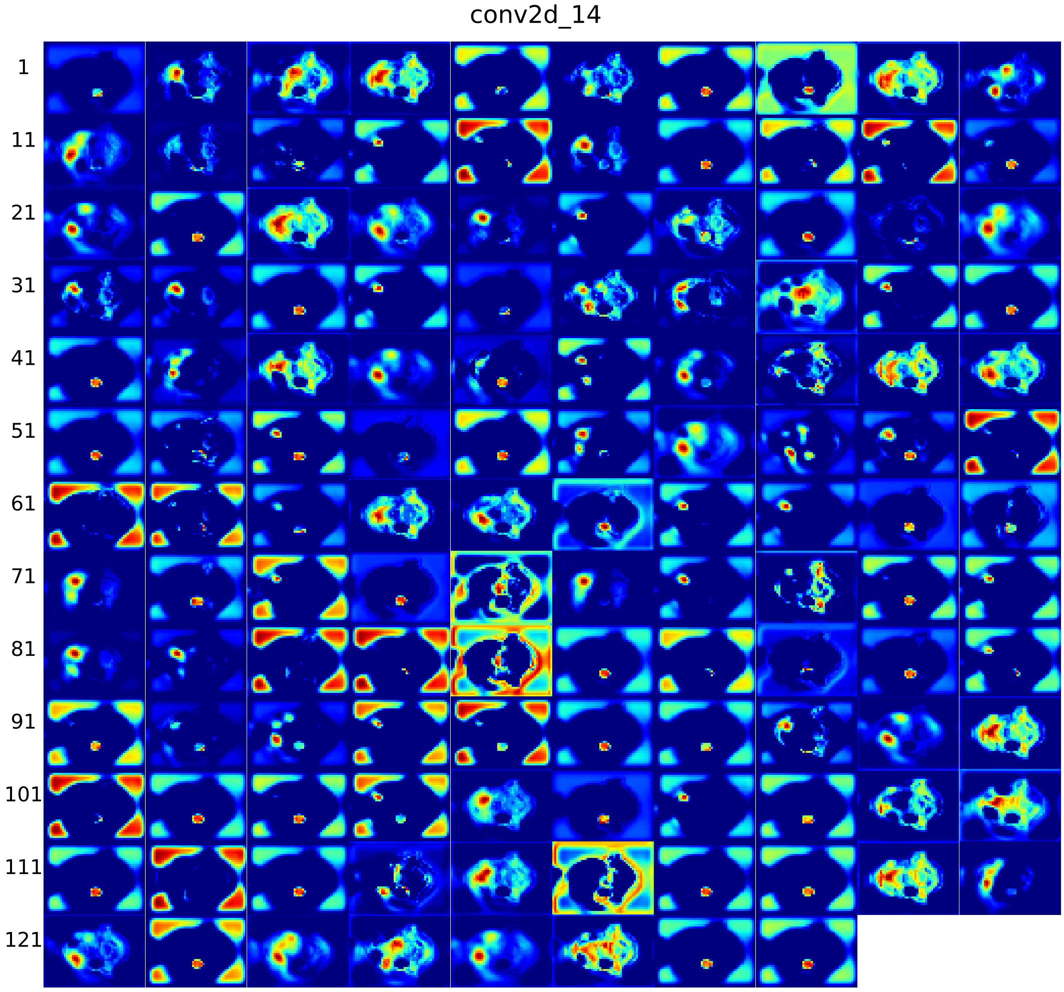
(m) Layer conv2d_12

Figure E.6: Activation maps at conv2d_12 layer (m) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



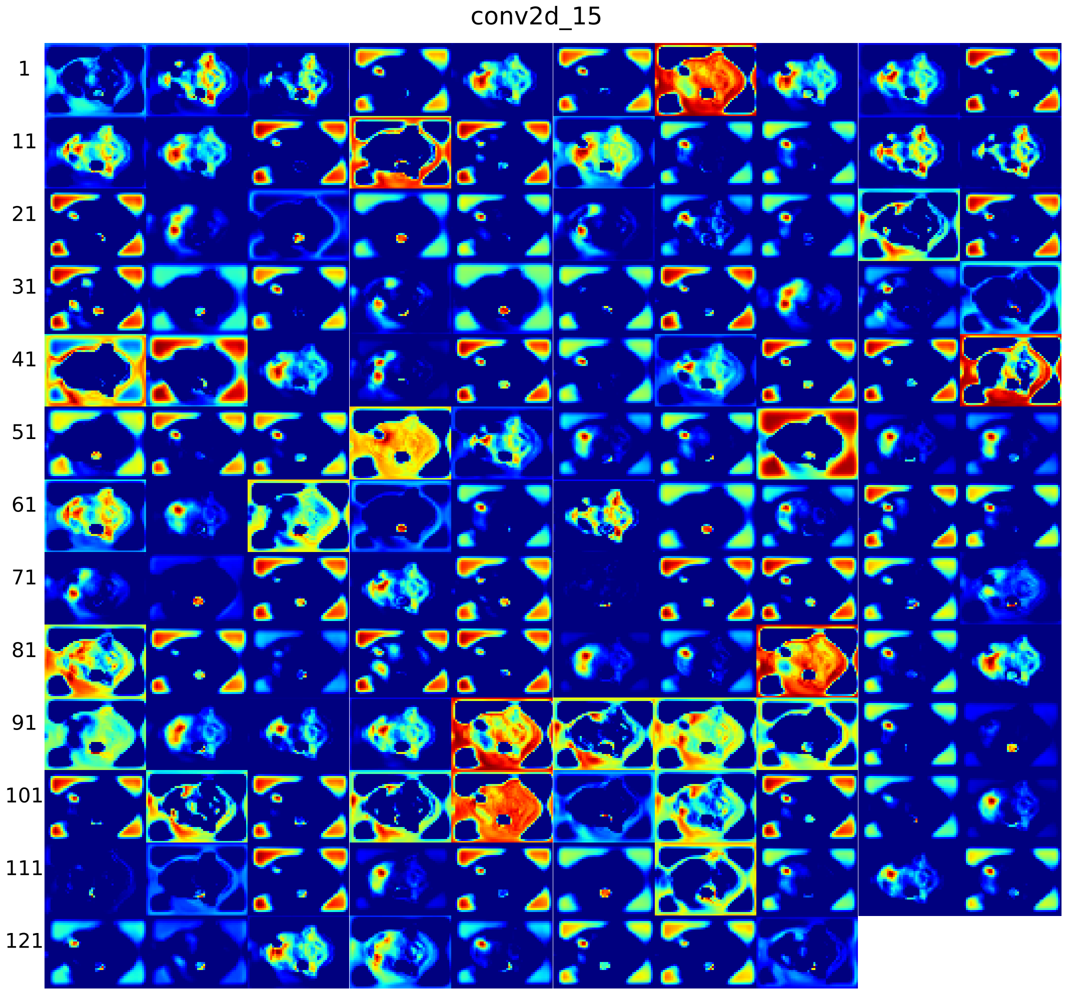
(n) Layer conv2d_13

Figure E.6: Activation maps at conv2d_13 layer (n) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



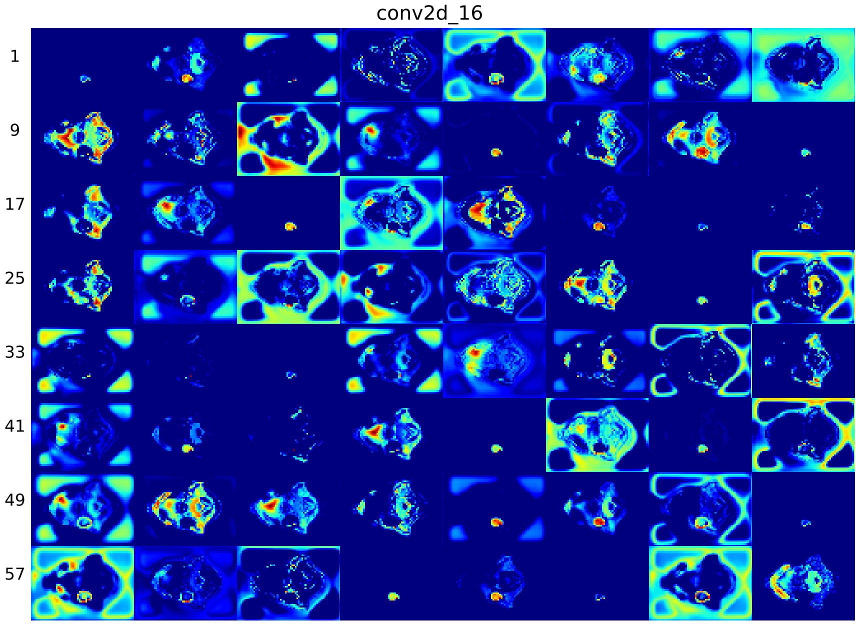
(o) Layer conv2d_14

Figure E.6: Activation maps at conv2d_14 layer (o) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.

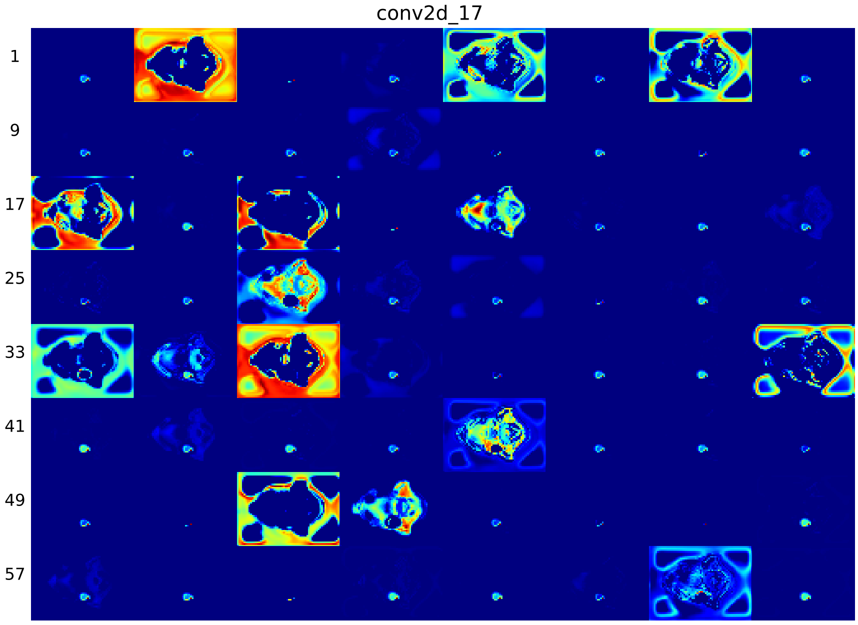


(p) Layer conv2d_15

Figure E.6: Activation maps at conv2d_15 layer (p) of image from patient 249, slice 55, with a Dice score of 0.95. Continued on next page.



(q) Layer conv2d_16



(r) Layer conv2d_17

Figure E.6: Activation maps at conv2d_16 layer (q) and conv2d_17 layer (r) of image from patient 249, slice 55, with a Dice score of 0.95.

Appendix F

Gradient-based visualization

All results generated from Section 4.2.3 on page 45 can be found in this part.

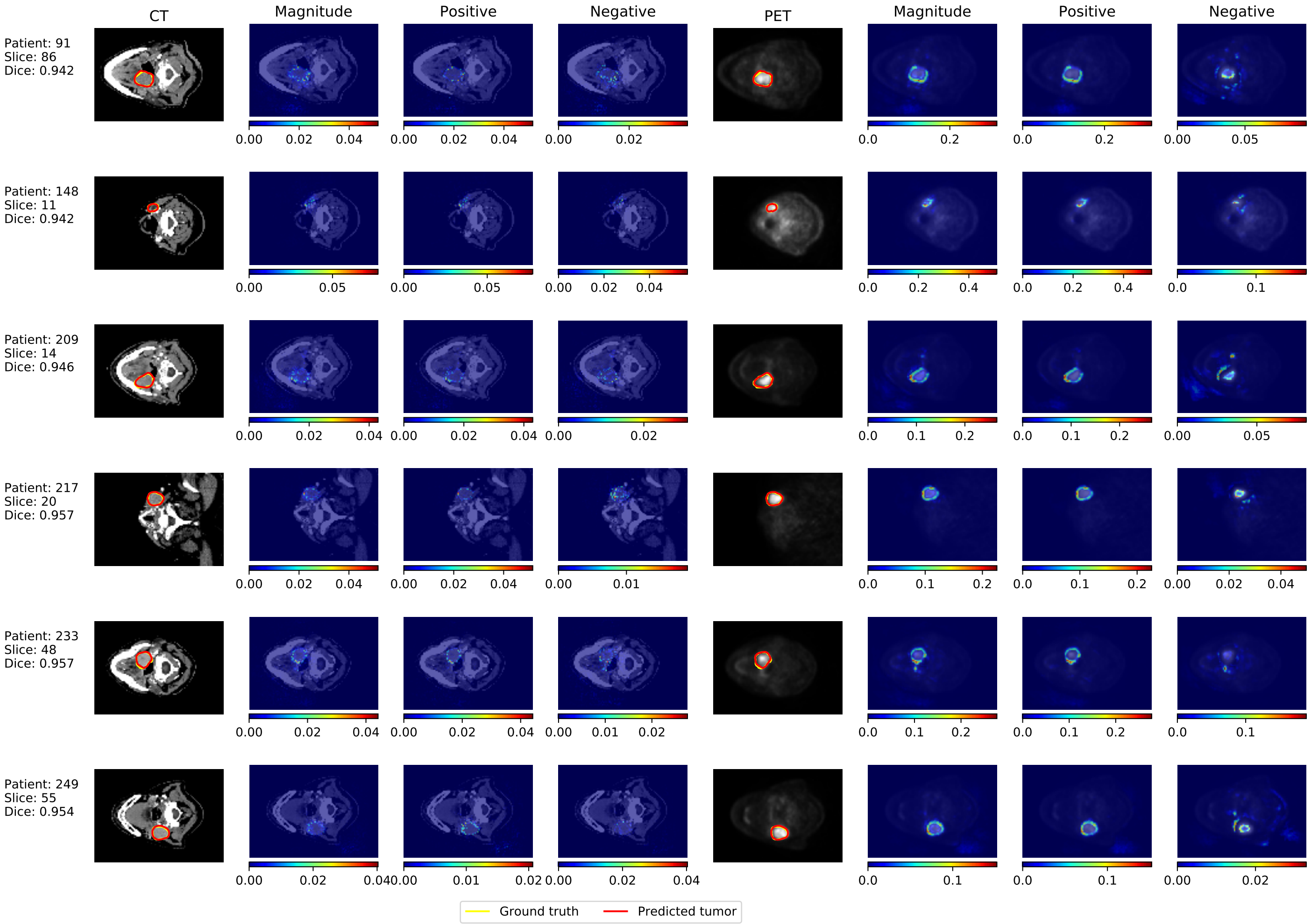


Figure F.1: Saliency maps using *positive prediction* loss function of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

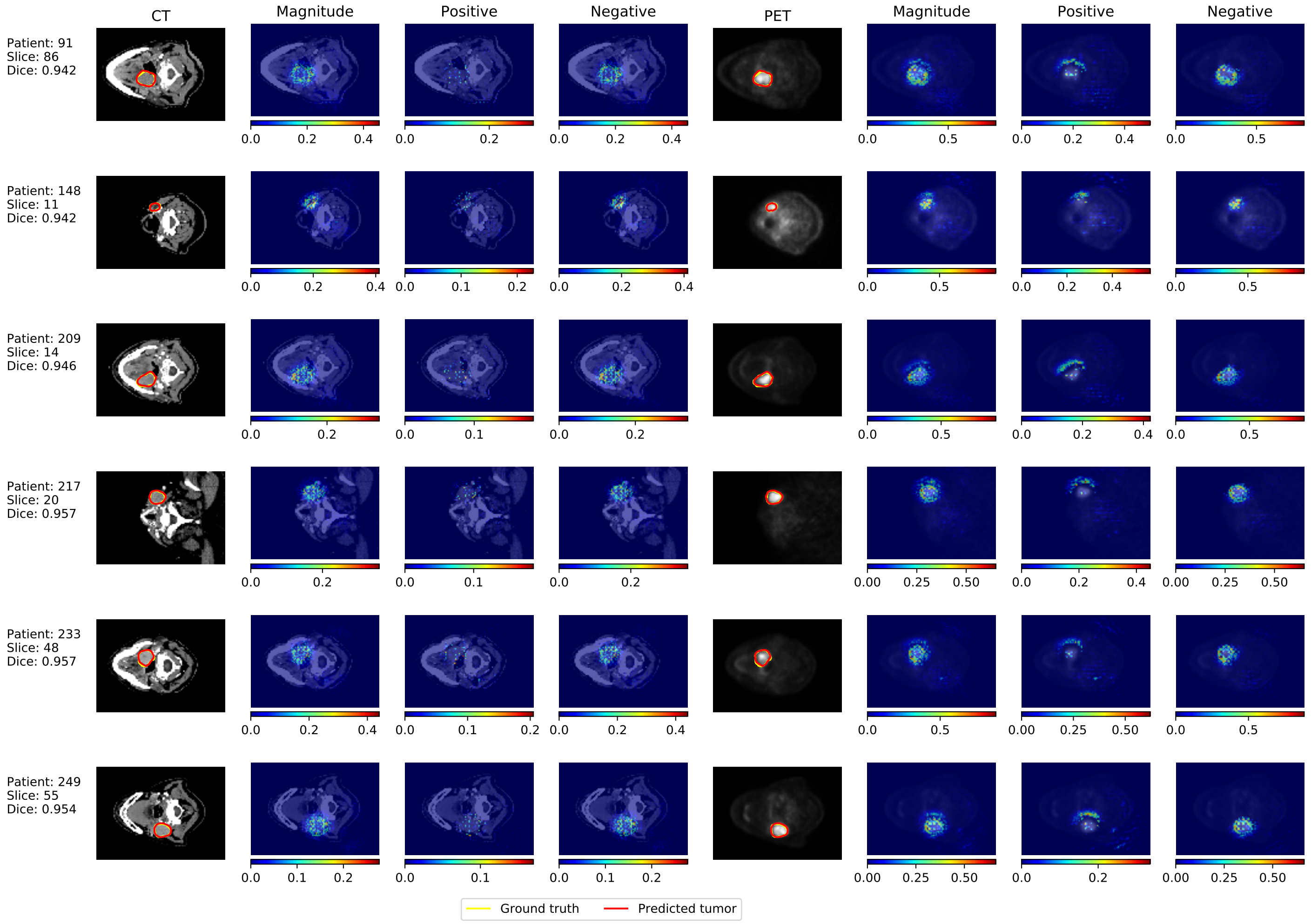


Figure F.2: Deconvnet-based visualization results using *positive prediction* loss function of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

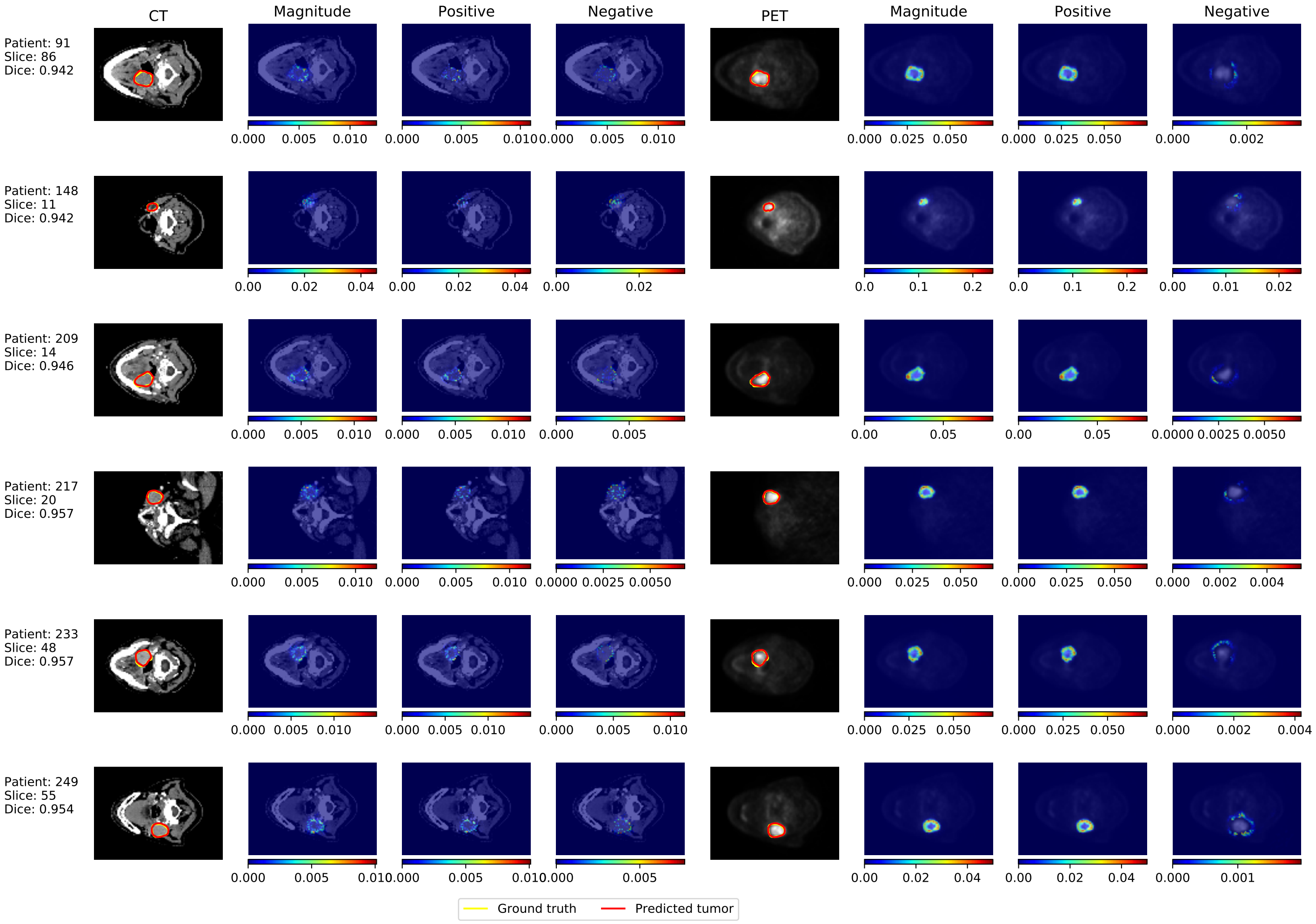


Figure F.3: Guided backpropagation-based visualization results using *positive prediction* loss function of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

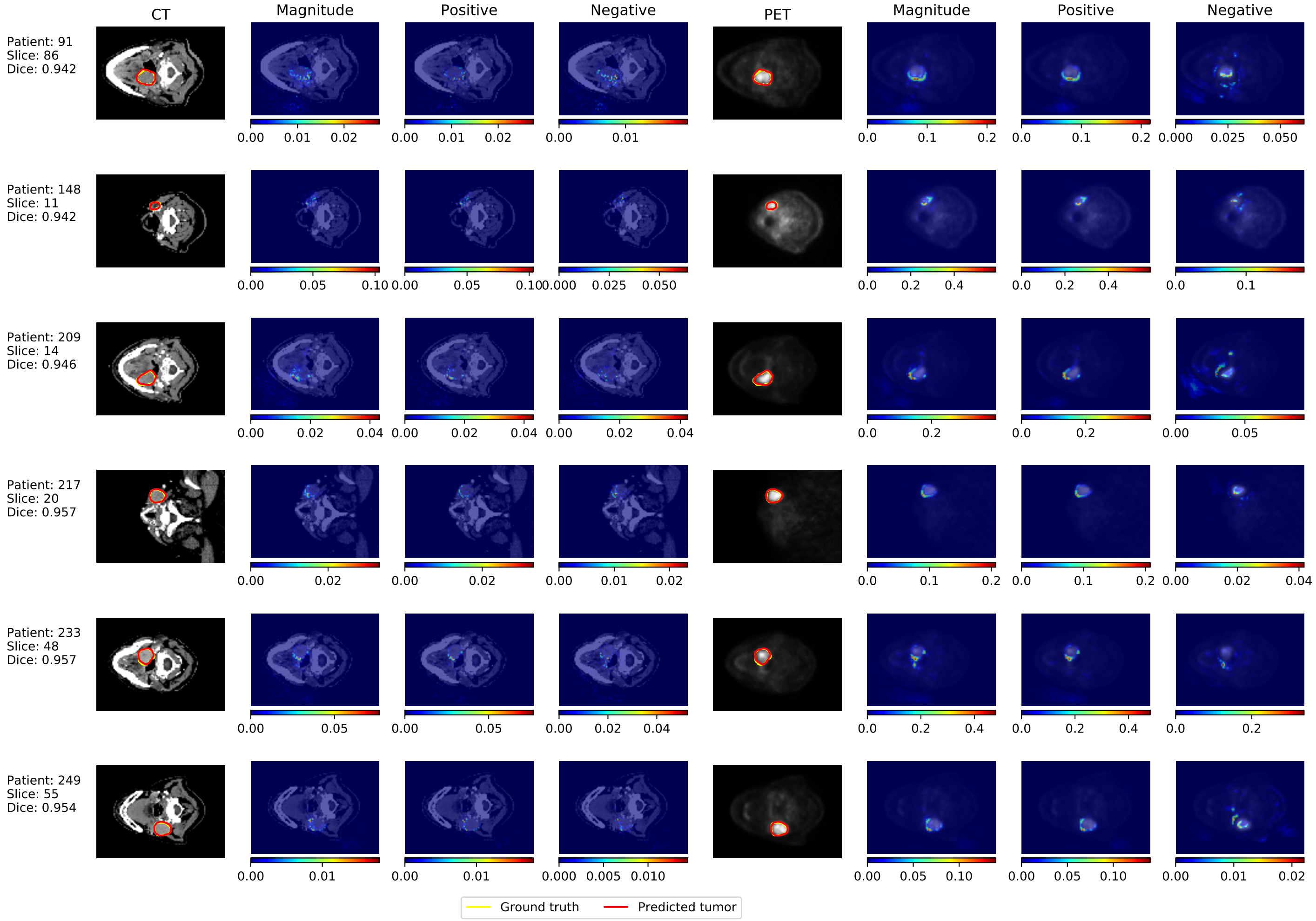


Figure F.4: Saliency maps using *true positive* loss function of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

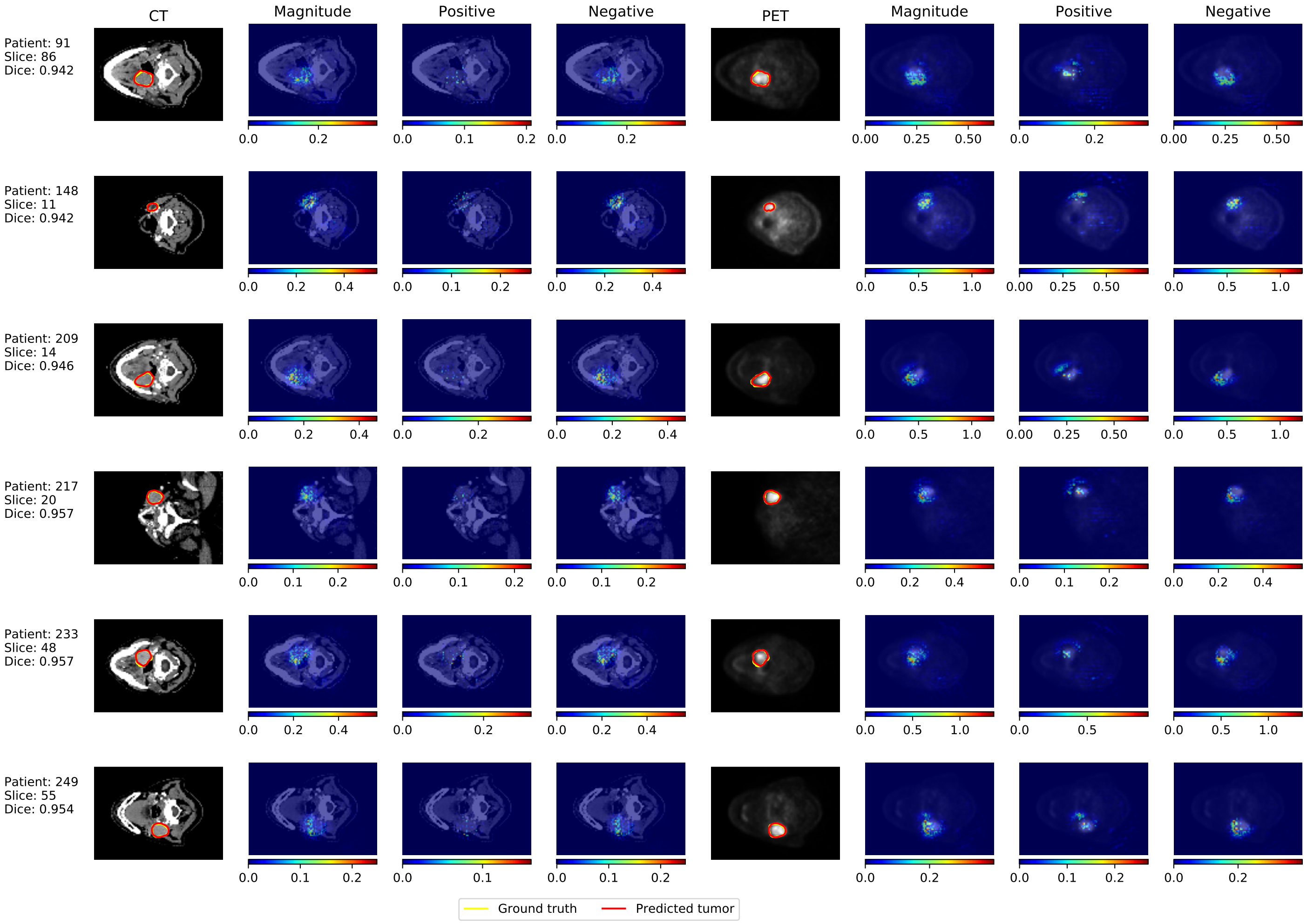


Figure F.5: Deconvnet-based visualization results using *true positive* loss function of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

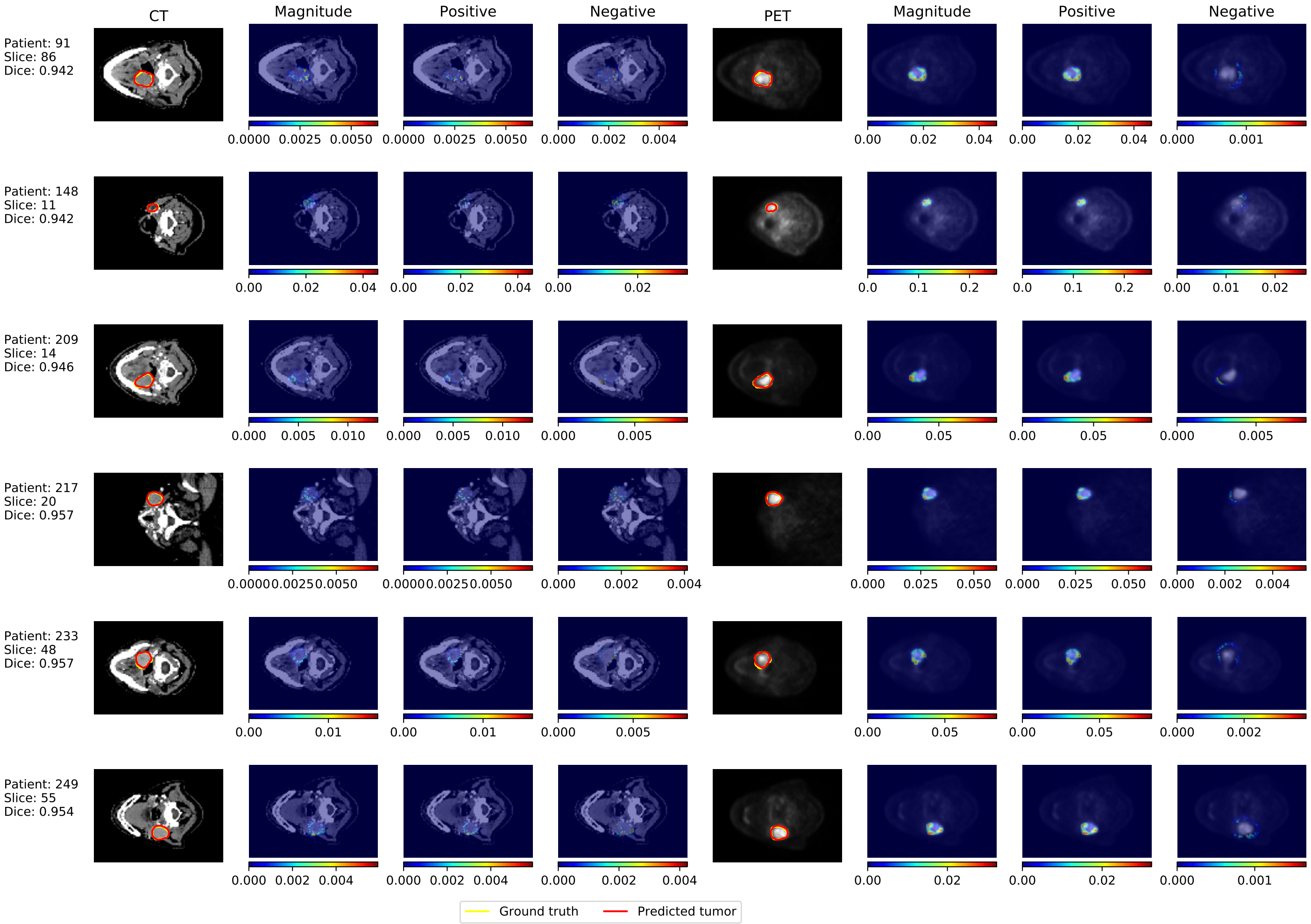


Figure F.6: Guided backpropagation-based visualization results using *true positive* loss function parts of images with high Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

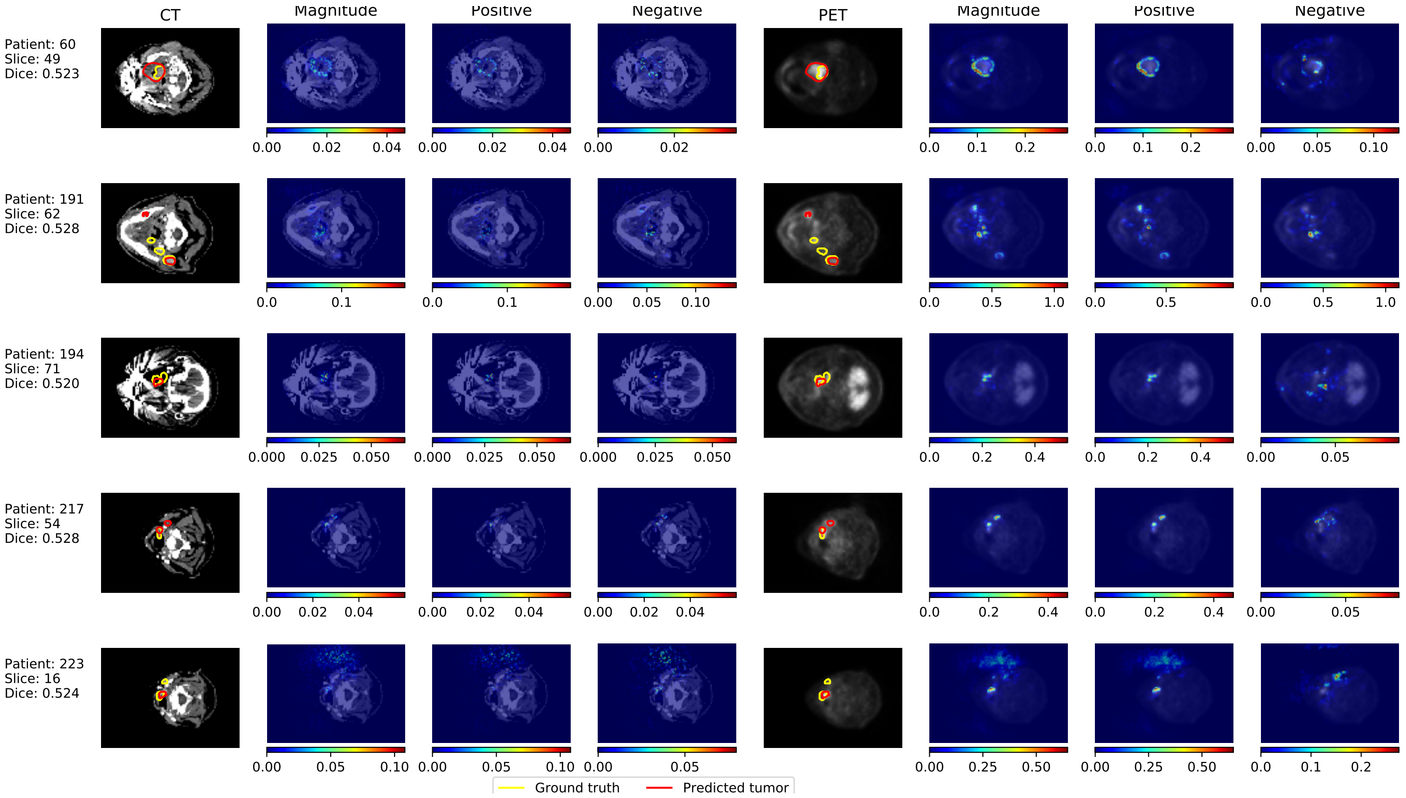


Figure F.7: Saliency maps using *positive prediction* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

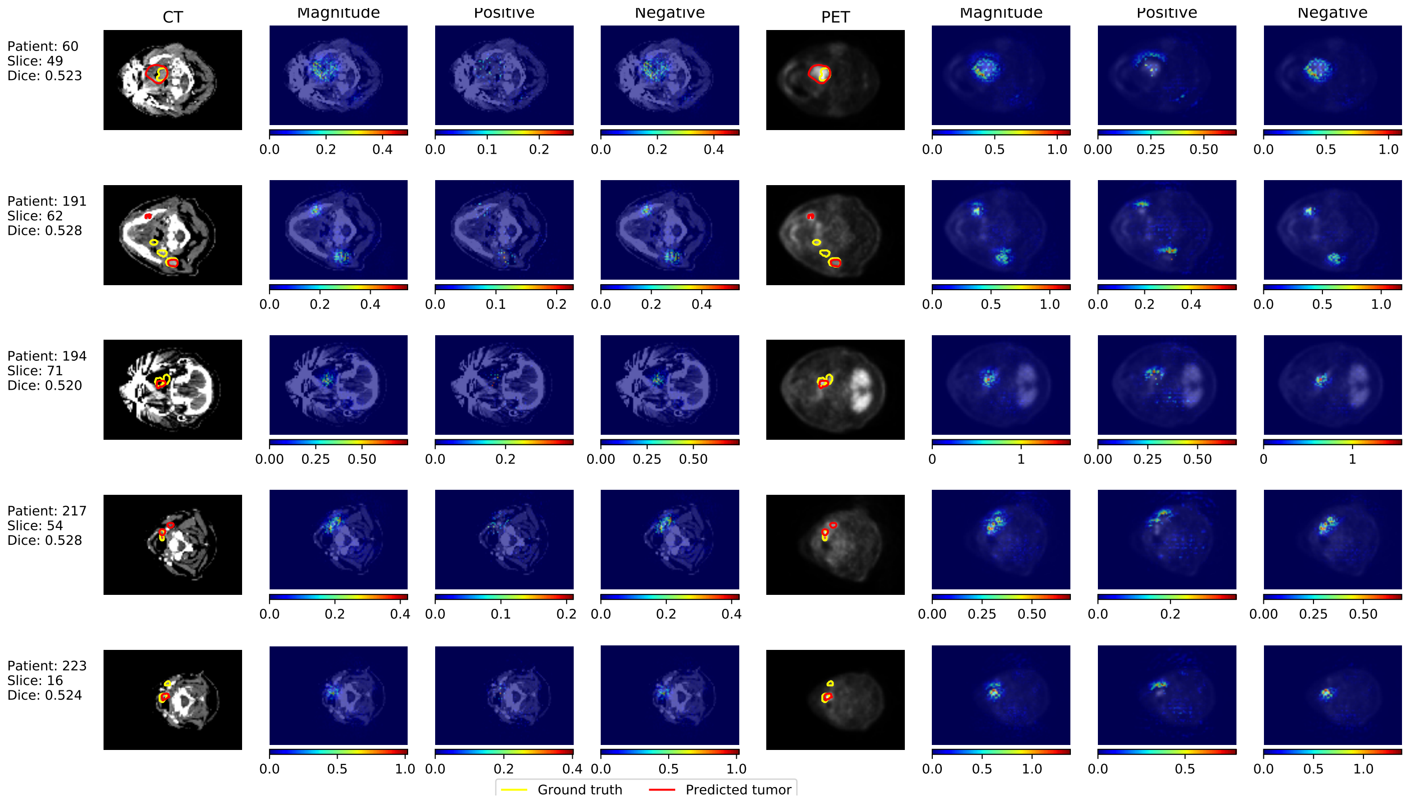


Figure F.8: Deconvnet-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

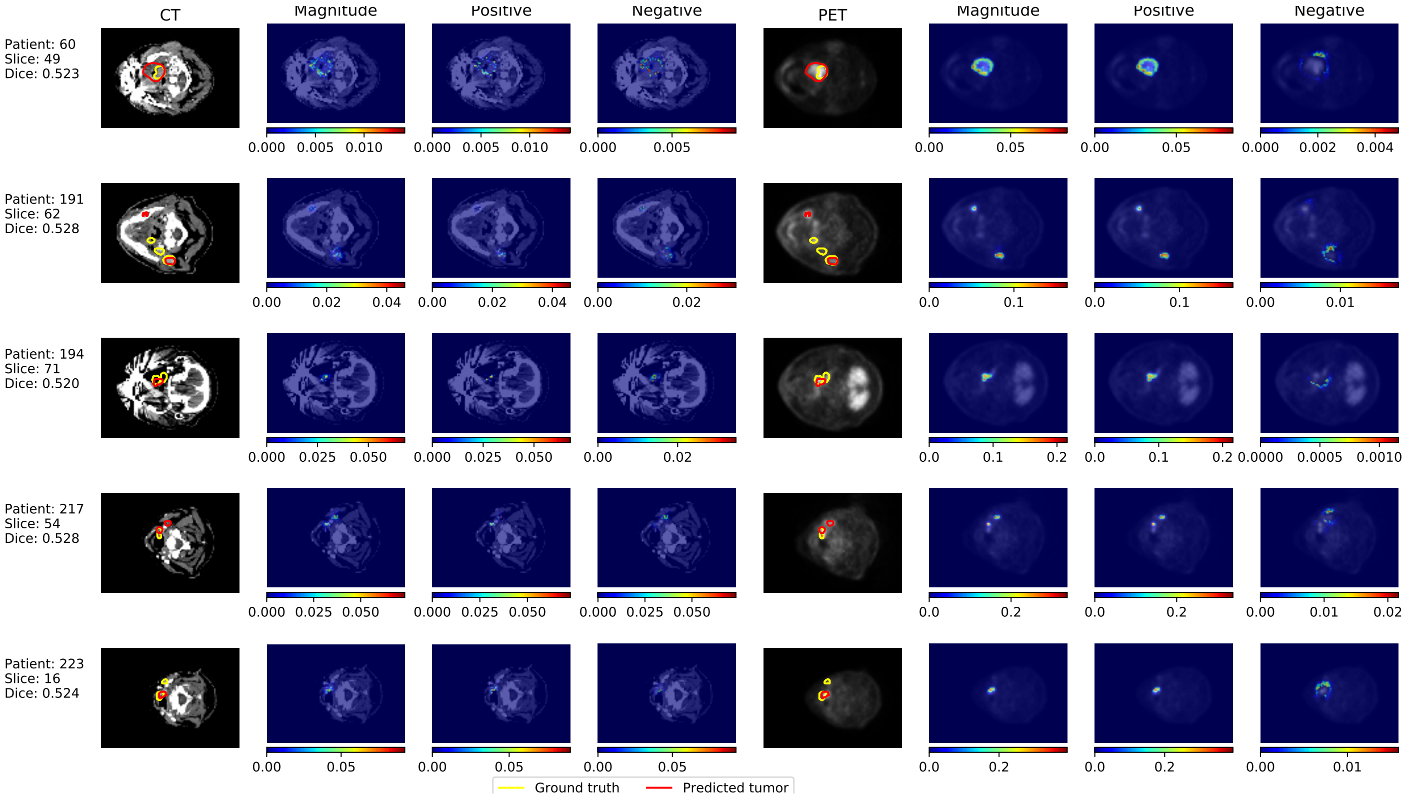


Figure F.9: Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

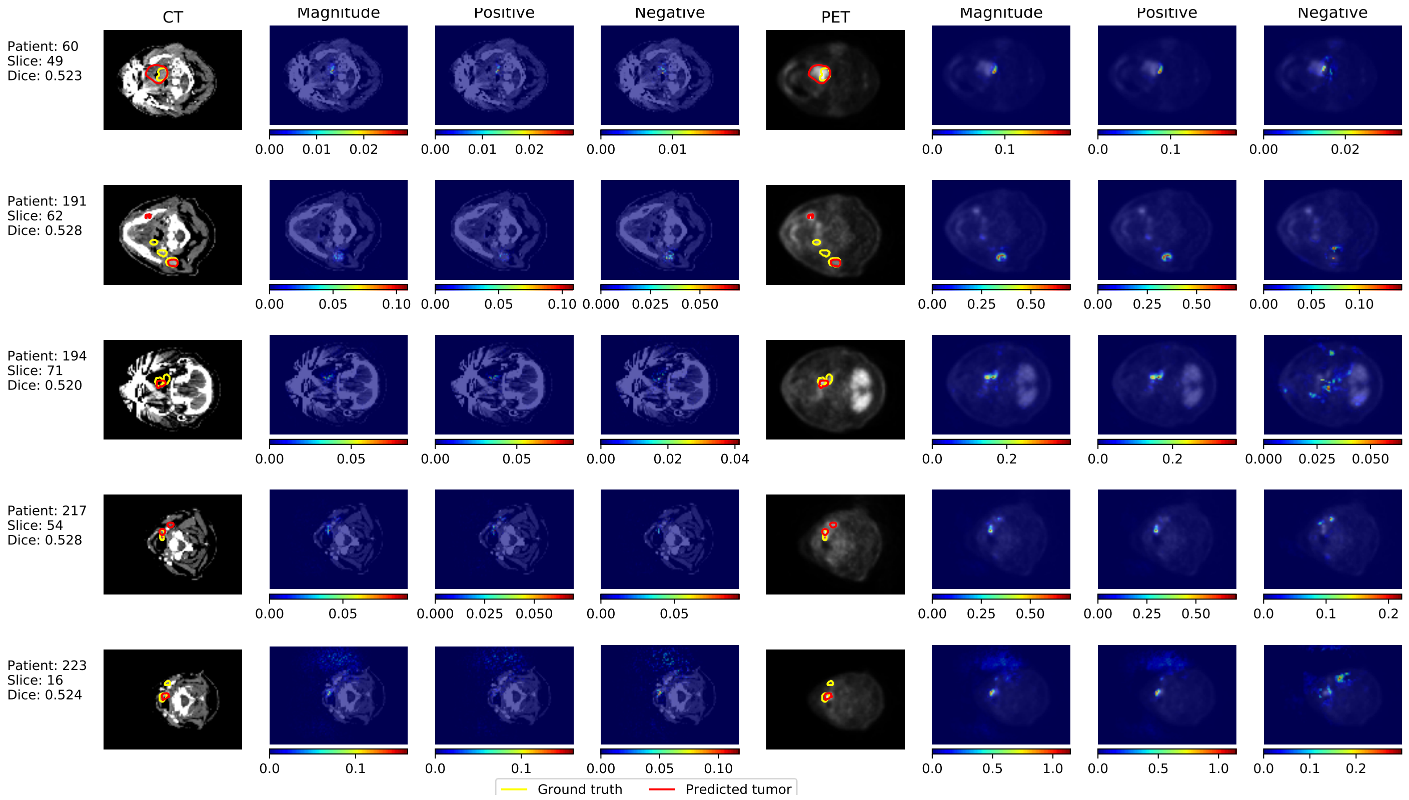


Figure F.10: Saliency maps using *true positive* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

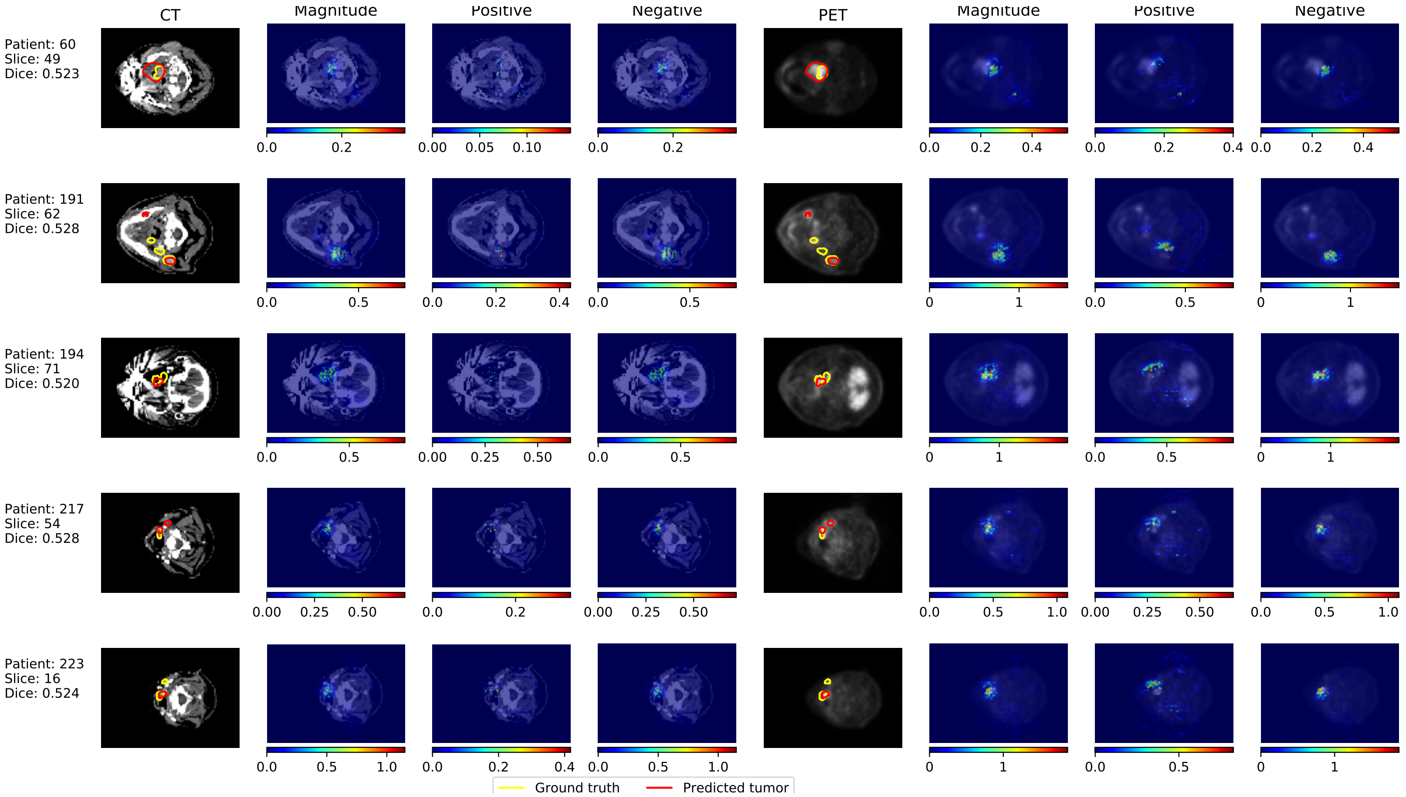


Figure F.11: Deconvnet-based visualization results using *true positive* loss function of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

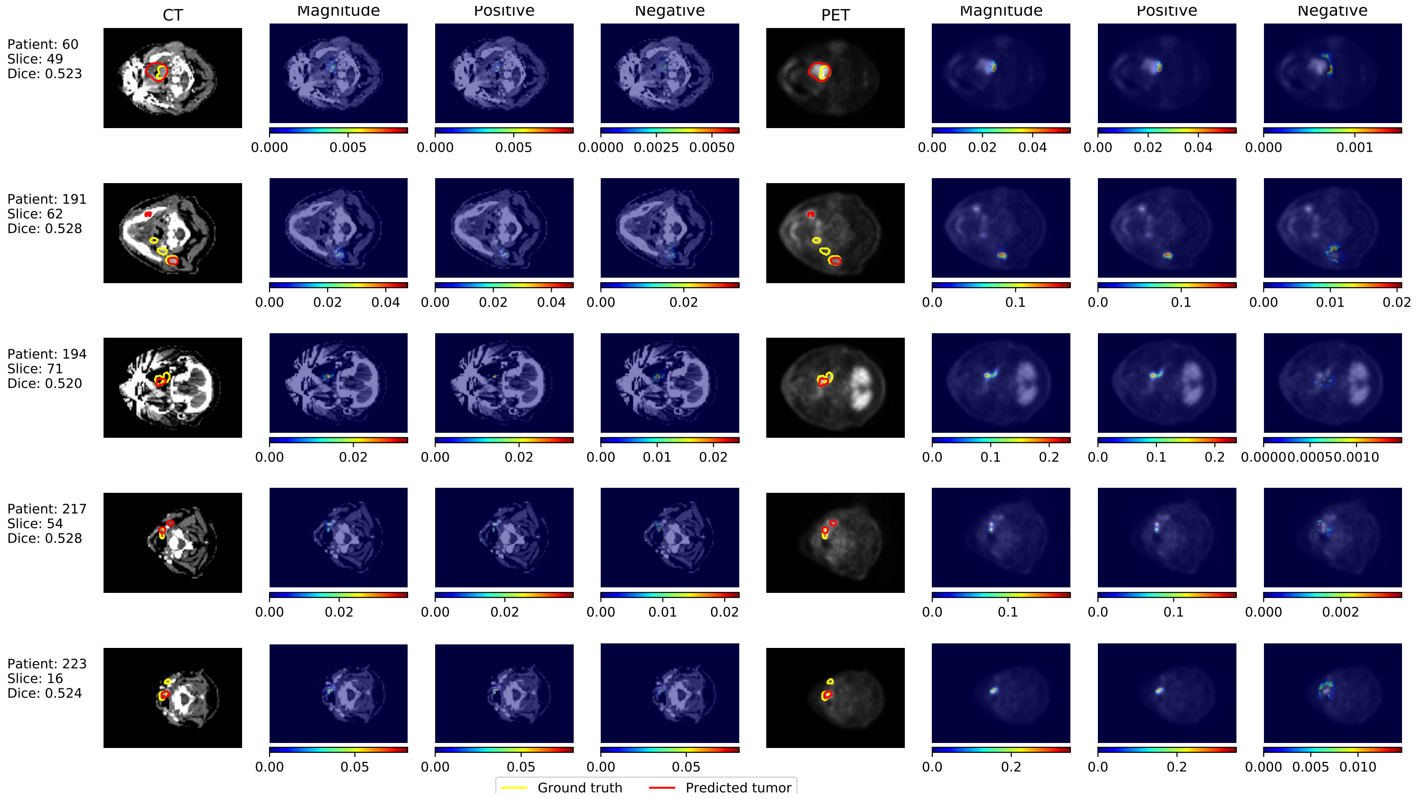


Figure F.12: Guided backpropagation-based visualization results using *true positive* loss function parts of images with 0.5 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

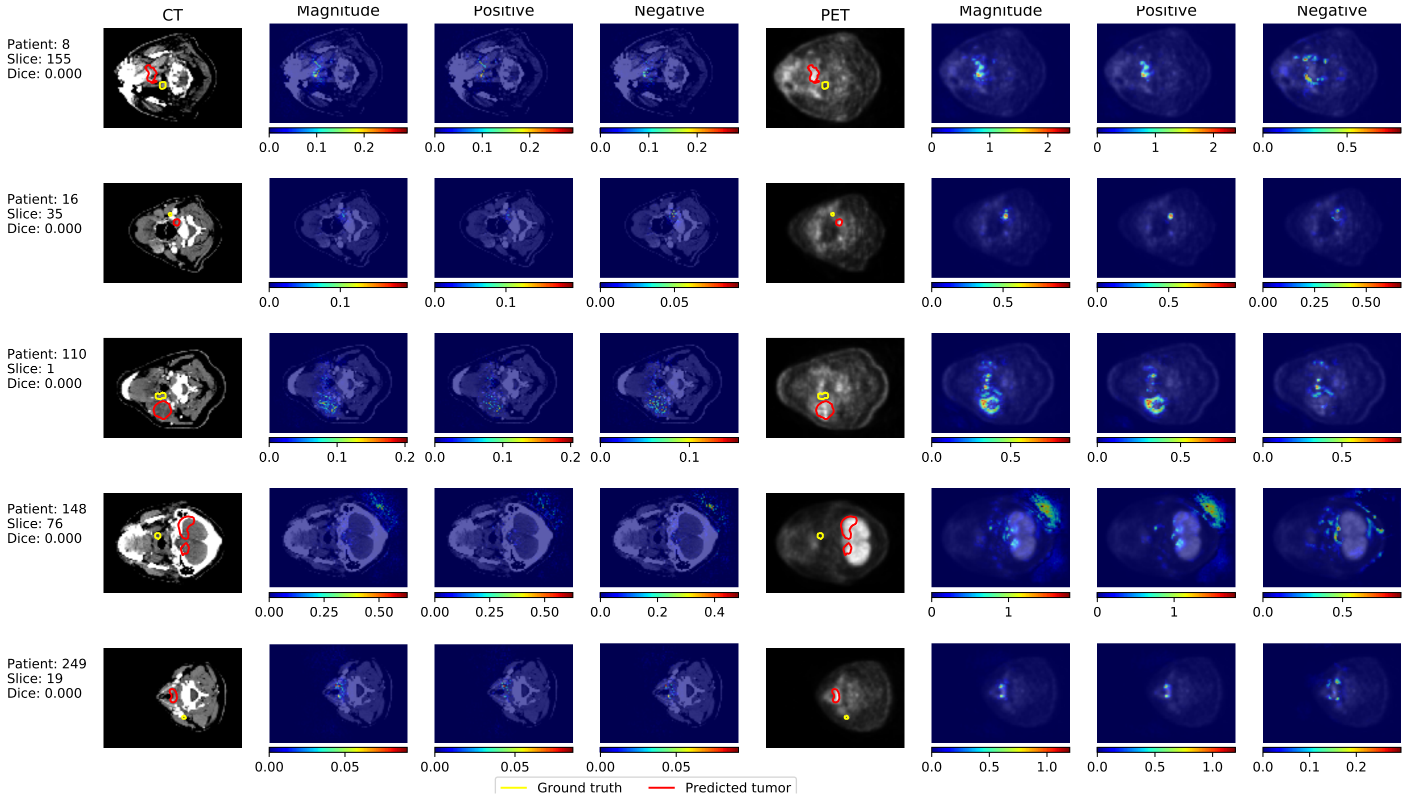


Figure F.13: Saliency maps using *positive prediction* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

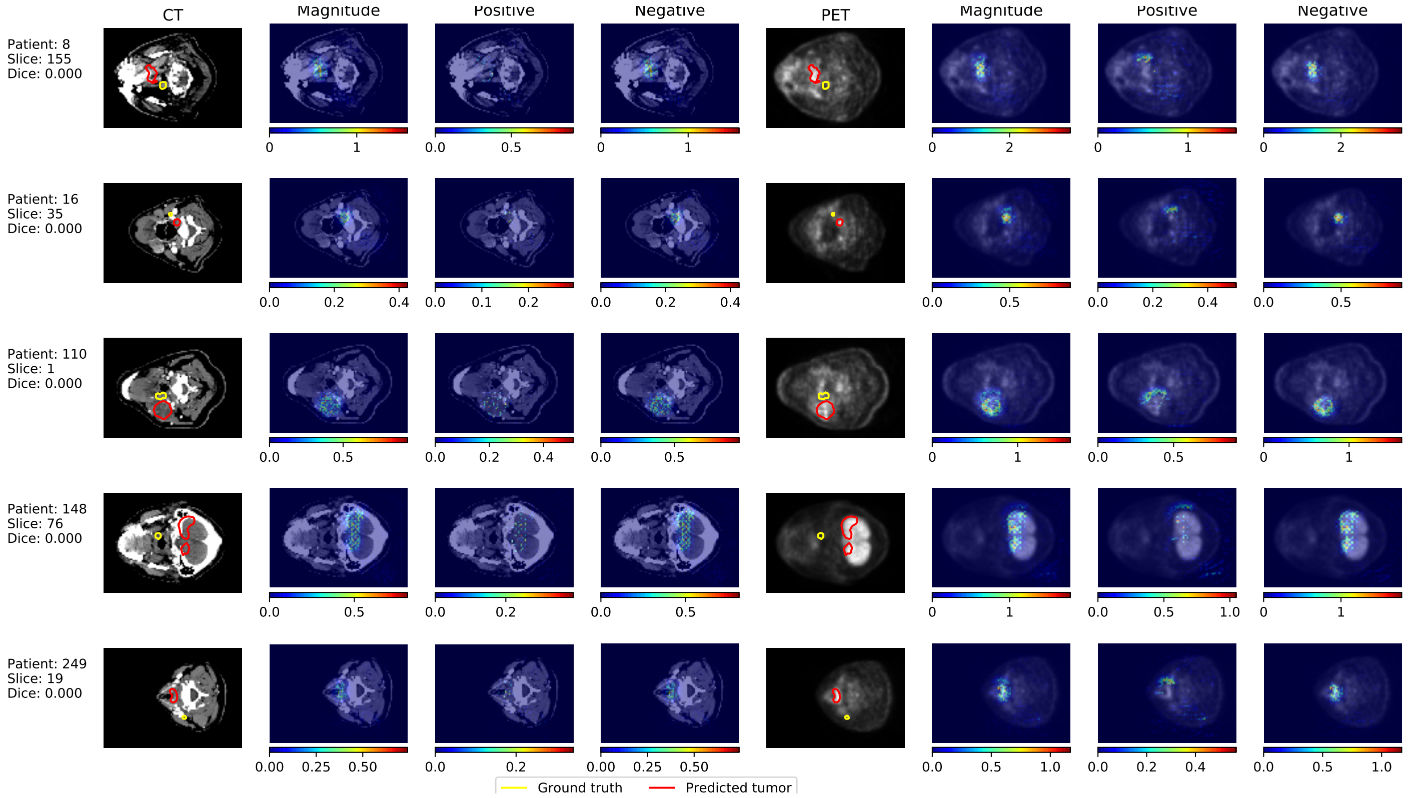


Figure F.14: Deconvnet-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

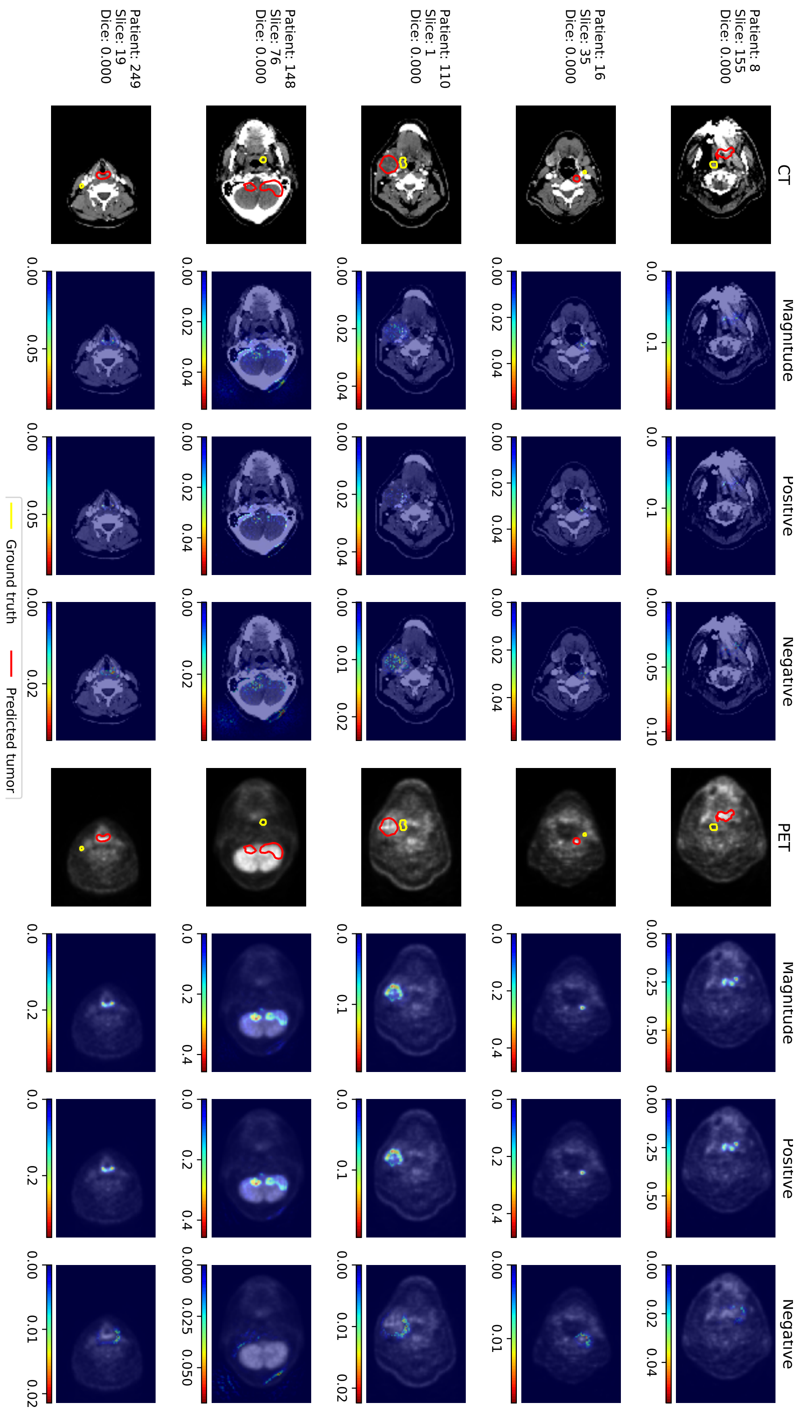


Figure F.15: Guided backpropagation-based visualization results using *positive prediction* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

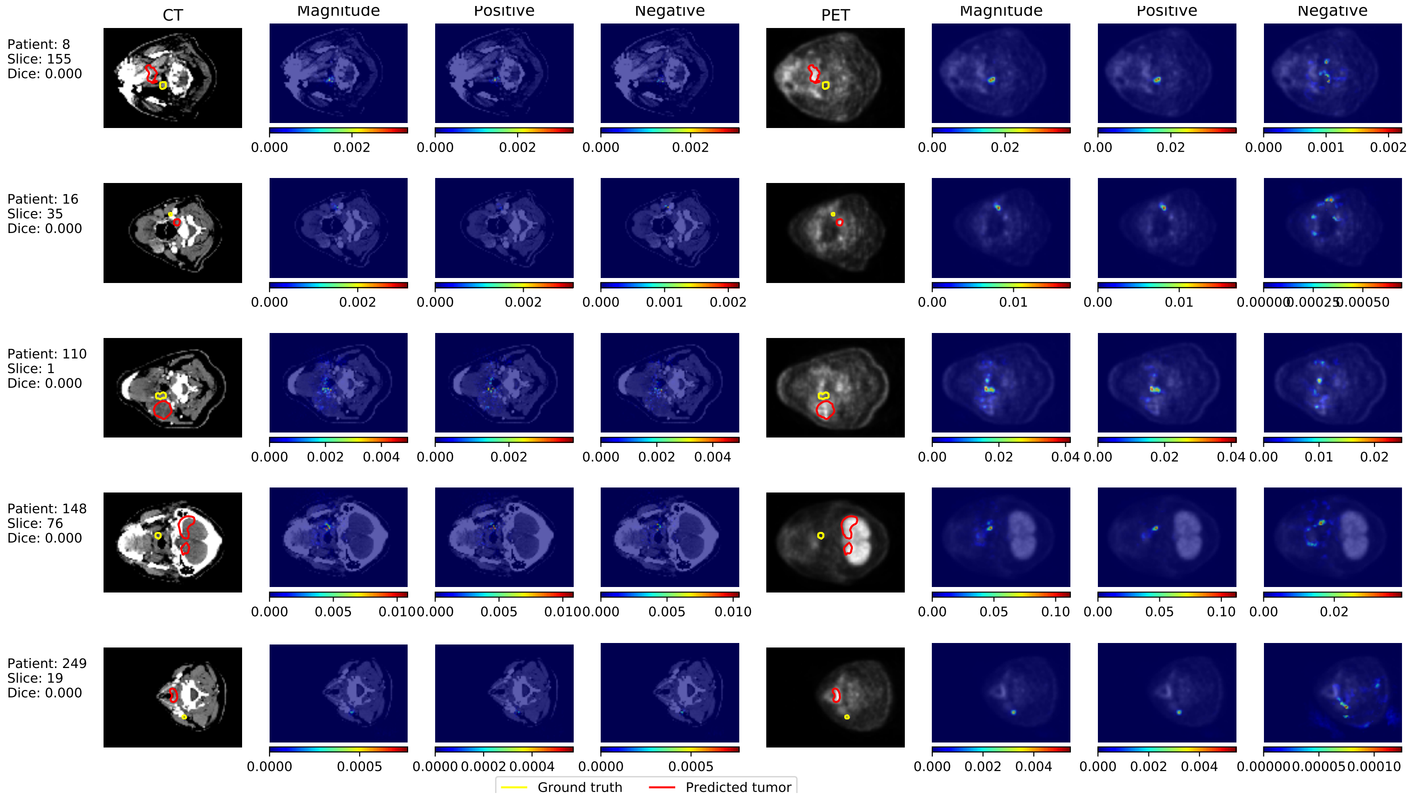


Figure F.16: Saliency maps using *true positive* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

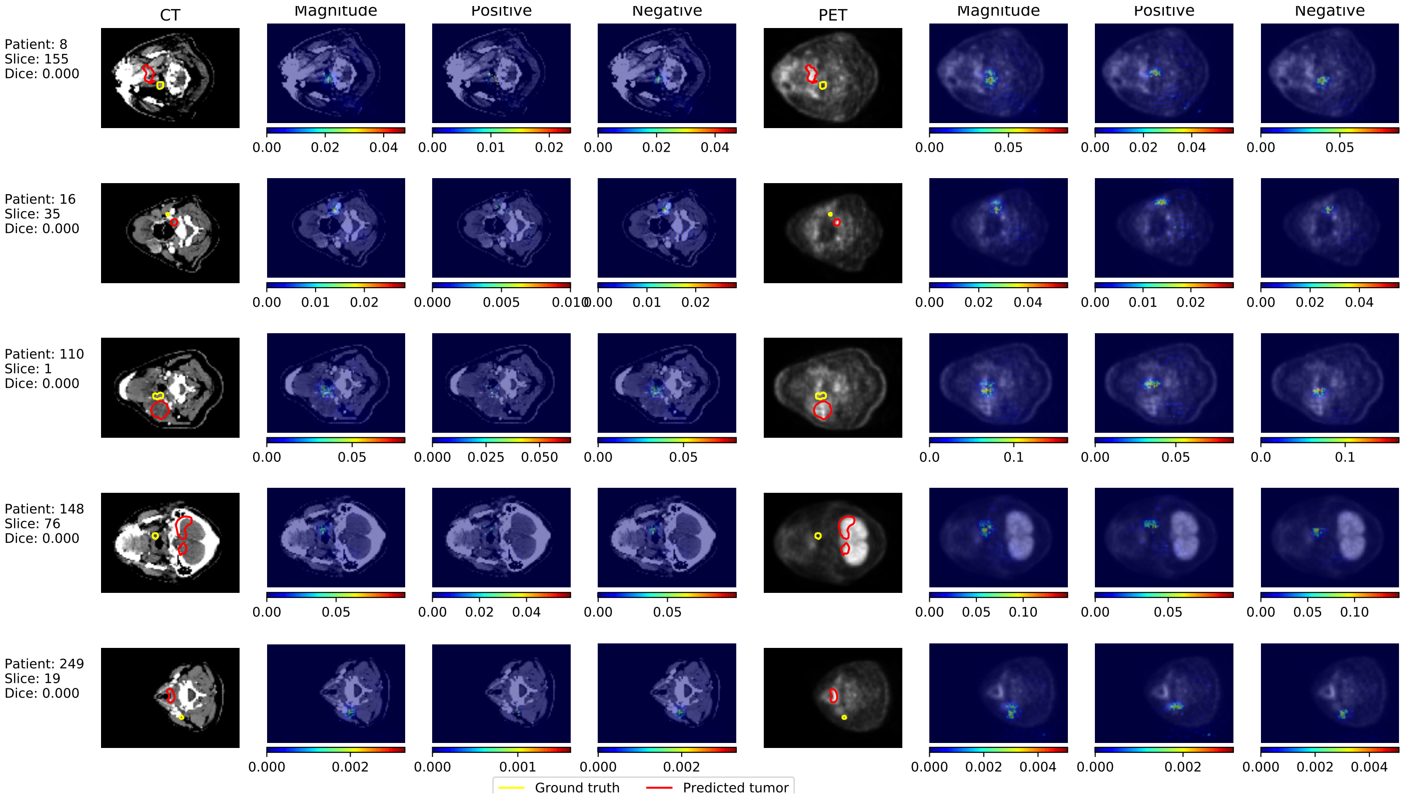


Figure F.17: Deconvnet-based visualization results using *true positive* loss function of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

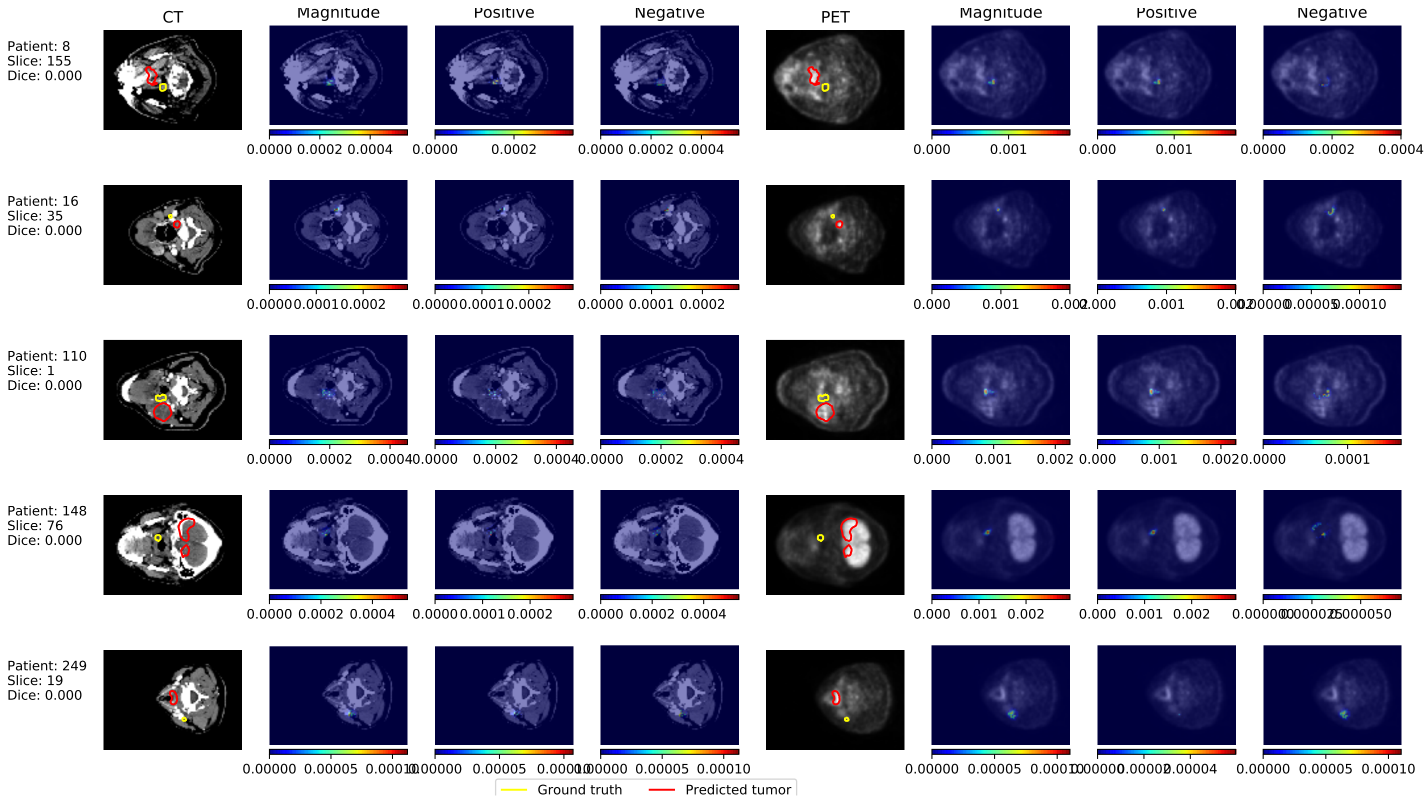


Figure F.18: Guided backpropagation-based visualization results using *true positive* loss function parts of images with 0.0 Dice scores. From left to right, the original CT scan, the magnitude of pixel importance on the CT scan, the positive magnitude of pixel importance on the CT scan, the negative magnitude of pixel importance on the CT scan, the original PET scan, the magnitude of pixel importance on the PET scan, the positive magnitude of pixel importance on the PET scan, the negative magnitude of pixel importance on the PET scan. The color bars show the level of pixel importance from low (left) to high (right) based on the gradient values.

Thank you.

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway