Eötvös Loránd University

Faculty of Informatics

Department of Programming Languages and Compilers

Histopathologic Cancer Detection: Identifying metastatic tissue in histopathologic slides using Deep Neural Networks

Supervisor:
Kitlei Róbert
assistant lecturer, MSc

Author:
Bauer Marko
Computer Science BSc

Budapest, 2020

Contents

1 Introduction **3**

1.1 Medical Image Analysis 3

1.2 Histopathologic Cancer Detection 4

1.3 Datasets 4

1.3.1 BreakHis Dataset 4

1.3.2 NCT-CRC-HE-100K Dataset 5

2 User Documentation **6**

2.1 System Requirements 6

2.1.1 General Software Requirements 7

2.1.2 Additional Software Requirements 7

2.2 Running the Program 8

2.3 Inspecting Datasets 9

2.4 Inspecting Networks 9

2.5 Basic Use: Predicting Tissue Type 10

2.6 Advanced Use: Visualizing Network Representations 11

2.6.1 Visualizing Heatmaps of Class Activations 11

2.6.2 Visualizing Intermediate Activations 12

2.6.3 Visualizing Filters of Convolutional Layers 12

2.7 Saving Results 13

3 Developer Documentation **14**

3.1 Program Structure 15

3.2 Use-Case Diagram 16

3.3 Class Diagrams 17

3.4 Creation of Datasets 19
3.5 CNNSimple Implementation 20
3.6 Transfer Learning 21
3.6.1 VGG19Simple Implementation 21
3.7 Experiments and Results 22
3.8 Graphical User Interface 24
3.8.1 Main Window 24
3.9 Utilities 25
3.10 Implementing Additional Features 25
3.11 Testing 26
3.12 External Code 26

4 Conclusion **27**
4.1 Future Work 27

A Convolutional Neural Networks **29**
A.1 Convolutional Layer 29
A.2 Max-Pooling Layer 30
A.3 Flatten Layer 30
A.4 Fully-Connected Layer 30
A.5 Dropout Layer 30

Bibliography **32**

Chapter 1

Introduction

Over the past decade deep learning has been reaching unimaginable heights, possibly starting another industrial revolution [1]. Numerous problems were solved with extraordinary accuracies, surpassing previous solutions, as well as human performance. Self-driving cars [2] and automated transportation [3], digital personal assistants [4], fraud detection [5], natural language processing [6] are just some of the tasks whose solutions use deep learning algorithms. But arguably the biggest impact was made in the field of computer vision [7], with image classification [8], object detection and segmentation [9], image colorization [10], reconstruction [11], super-resolution [12] and synthesis [13], and as a result, the entire healthcare industry is on the verge of a major transformation. Deep learning algorithms could be used in various ways, such as analyzing electronic health records [14], detecting heart problems [15], diagnosing cancer [16], planning, navigating and performing surgery [17].

1.1 Medical Image Analysis

The majority of the data present in medicine (over 90% of it) belongs to the imaging data, and it is natural to assume that deep learning could have huge impact on the future of medicine. Computed tomography (CT), magnetic resonance imaging (MRI), and X-rays are just some of the medical imaging techniques which produce data that can be used in deep learning algorithms. In this paper I focus on analysis of histopathology slides, microscopic images of tissue obtained during the biopsy.

1.2 Histopathologic Cancer Detection

Histopathologic cancer detection refers to the problem of detecting cancerous tumors in pathologic scans of lymph nodes, organs of the lymphatic system (widely present throughout the body), which act as filters for foreign particles and cancer cells. This task is clinically-relevant, as pathologists analyze a great number of histopathologic slides on a daily basis, which is a time-consuming and tedious task prone to misinterpretation. Hence a lucrative solution would reduce their workload, speed up the process, and improve the prediction accuracy. In this paper, I propose a deep learning algorithm to solve this task.

1.3 Datasets

Digital pathology is a field in medical imaging, where whole-slide scanners are used to digitize glass slides containing tissue at high resolution, in order to be viewed, managed, shared, and analyzed on a computer. The advance of digital pathology has led to the high availability of digital images, which has in turn led to the creation of datasets on which deep learning methods can be applied. In this paper, I work with two such datasets, BreakHis and NCT-CRC-HE-100K.

1.3.1 BreakHis Dataset

The Breast Cancer Histopathological Image Classification [18] (BreakHis) is a dataset constructed in collaboration with Pathological Anatomy and Cytopathology Laboratory (Parana, Brazil) which contains 9.109 images of breast tumor tissue. It is collected from 82 patients, using different magnifying factors (40x, 100x, 200x, and 400x), and contains 2.480 benign and 5.429 malignant samples (700×460 pixels). Dataset is divided into two main groups, benign tumors (do not invade nearby tissue or spread to other parts of the body) and malignant tumors (can invade nearby tissues; made of cancer cells). Based on the way cells look under the microscope, there are eight subtypes of tumors: adenosis, fibroadenoma, phyllodes tumor and tubular adenoma for benign tumors, and ductal carcinoma, lobular carcinoma, mucinous carcinoma and papillary carcinoma for malignant tumors (Figure 1.1).

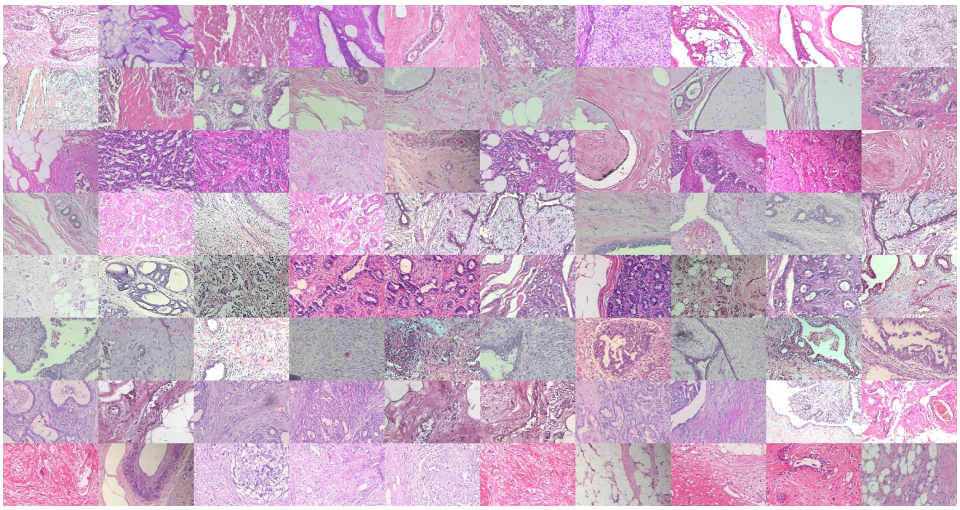


Figure 1.1: BreakHis dataset tumor types: adenosis, ductal carcinoma, fibroadenoma, lobular carcinoma, mucinous carcinoma, papillary carcinoma, phyllodes tumor, tubular adenoma

1.3.2 NCT-CRC-HE-100K Dataset

The NCT-CRC-HE-100K [19] is a dataset constructed in collaboration with the National Center for Tumor diseases (Heidelberg, Germany) which contains 100.000 images of colorectal tissue. It is collected from 86 patients, using a 100x magnifying factor (224×224 pixels). Based on the way cells look under the microscope, there are nine subtypes of tissue: adipose, background, debris, lymphocytes, mucus, smooth muscle, normal colon mucosa, cancer-associated stroma, and colorectal adenocarcinoma epithelium (Figure 1.2).

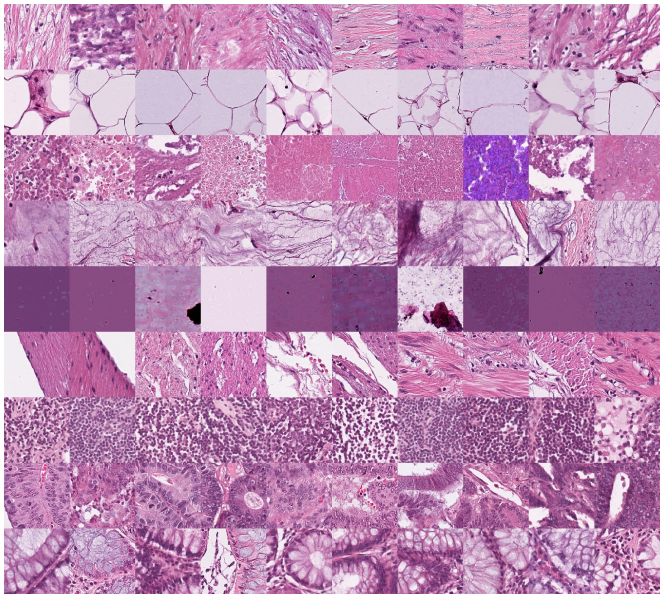


Figure 1.2: NCT-CRC-HE-100K dataset tissue types: adipose tissue, background, cancer-associated stroma, colorectal adenocarcinoma epithelium, debris, lymphocytes, mucus, normal colon mucosa, smooth muscle

Chapter 2

User Documentation

There are three main ways of using Histopathologic Cancer Detection program:

1. Predicting tissue and cancer subtype of breast and colorectal tissue, i.e. predicting whether the patient has breast or colorectal cancer or not
2. Visualizing what Convolutional Neural Network (CNN) learns, i.e. visualizing how network transforms input image, and which parts of input image lead to predicting tissue and cancer subtype
3. Adding new datasets and networks in order to increase the scope of the program, and predict even more tissue and cancer subtypes

Before using the program, certain system requirements must be satisfied.

2.1 System Requirements

In order to run Histopathologic Cancer Detection, Python 3.5+ is required on one of the following operating systems: Windows 7 or later (64bit), macOS 10.12.6 (Sierra), or later (64bit), Ubuntu 16.04 or later (64bit) or Raspbian 9.0 or later. Moreover, considering computer needs to be powerful enough to perform a large number of tensor operations, CPU Intel Core i5 6th generation processor or higher (or an AMD equivalent processor) and 8+GB of RAM are required.

2.1.1 General Software Requirements

In order to use the program to predict tissue and cancer subtype, and to visualize what CNNs learn, following Python dependencies are required:

- `H5py` - interface to the HDF5 binary data format, which can store huge amounts of numerical data, and easily manipulate that data from `NumPy` (used to save and load network weights)
- `Keras`, `Keras-Applications`, `Keras-Preprocessing`, `TensorFlow` - neural network APIs, which can build and deploy machine learning applications (used to build and train neural networks)
- `Matplotlib`, `seaborn` - data visualization libraries (used for visualization of datasets, neural networks and tissue and cancer subtype predictions)
- `NumPy` - library which provides support for large, multi-dimensional arrays
- `OpenCV-Python`, `Pillow`, `scikit-image` - image processing libraries (used to load, process and save images)
- `pandas` - data analysis and manipulation library (used for loading datasets)
- `PyQt5` - python binding of cross-platform GUI toolkit Qt, which contains substantial set of GUI widgets (used for implementing GUI of the program)
- `scikit-learn` - machine learning library for predictive data analysis (used for its metrics in order to assess network performance)

2.1.2 Additional Software Requirements

In order to replicate results of already existing networks, or to expand the scope of the program by adding new datasets and training new networks, NVIDIA GPU card with CUDA Compute Capability 3.5 or higher is required, along with the following:

- NVIDIA CUDA toolkit - provides a development environment for creating high-performance GPU-accelerated applications
- NVIDIA CUDA Deep Neural Network Library (cuDNN) - GPU-accelerated library of primitives for deep neural networks, which provides highly tuned implementations for standard routines

- `TensorFlow-GPU` - GPU-enabled version of `TensorFlow` library

Training convolutional neural networks requires a large amount of computations, and in order to decrease training time, networks are trained on GPU (it is not feasible on CPU, as GPU training is multiple times faster).

2.2 Running the Program

After running the program, new window appears (Figure 2.1).

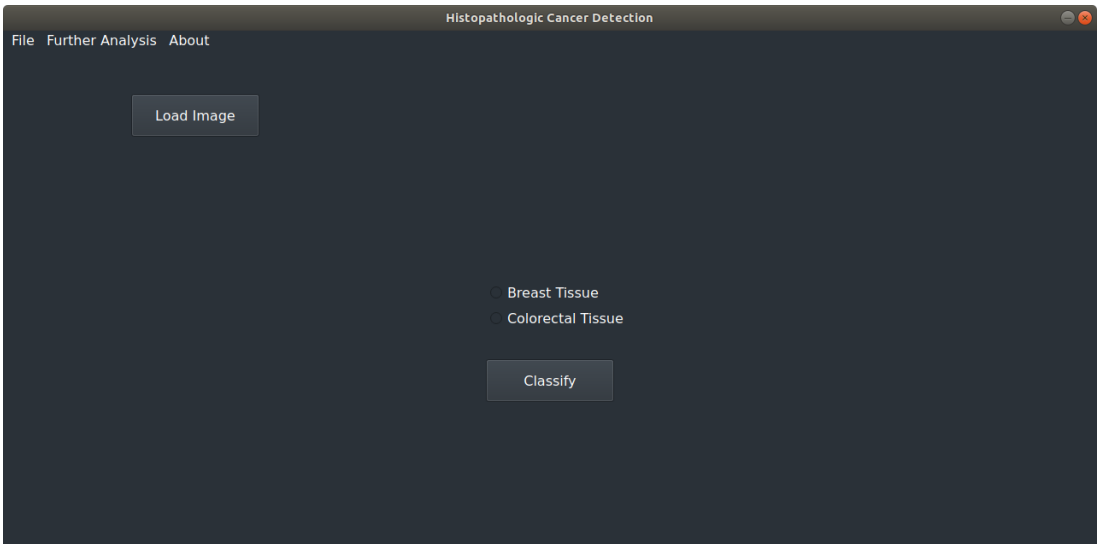


Figure 2.1: The main window of Histopathologic Cancer Detection program

From here, it is possible to:

1. Inspect datasets (Section 2.3) and networks (Section 2.4)
2. Load histopathologic slide of breast or colorectal tissue and predict the tissue and cancer subtype (Section 2.5)
3. Further visualize network representations by analyzing heatmaps of class activations, filters of convolutional layers and intermediate activations (Section 2.6)

2.3 Inspecting Datasets

In order to find basic information about datasets, which include sample images from dataset, tissue and cancer subtypes, as well as number of images per category, go to *About* → *About Datasets* and choose dataset (Figure 2.2). In current implementation there are two datasets available: BreakHis and NCT-CRC-HE-100K.

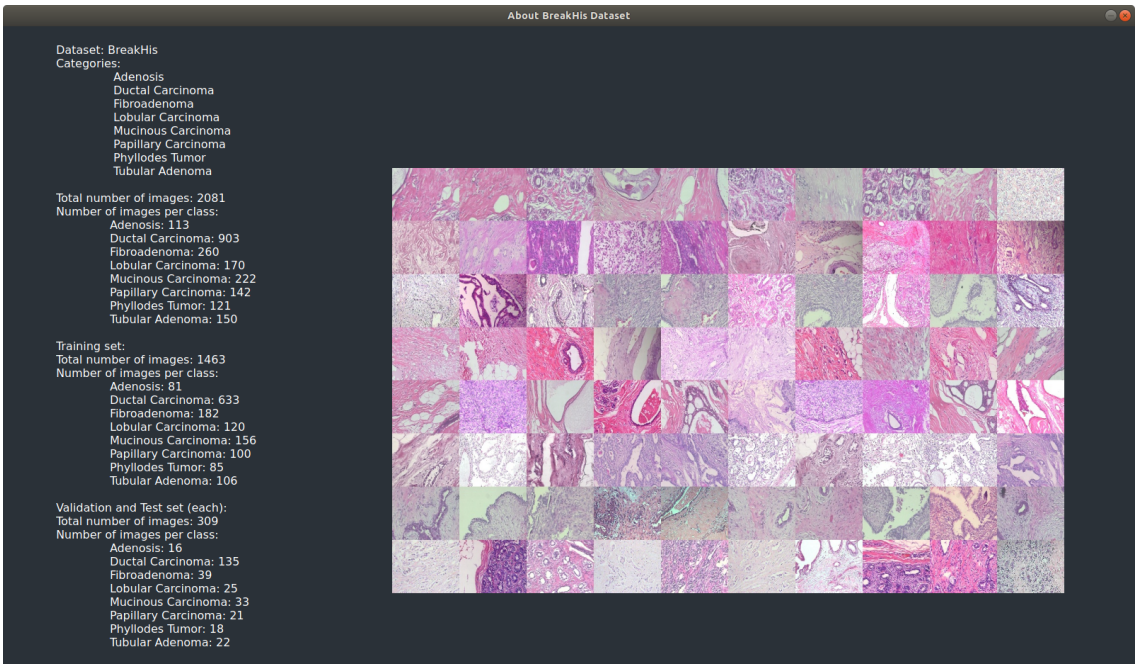


Figure 2.2: BreakHis dataset basic information

2.4 Inspecting Networks

In order to find basic information about networks, which include network architecture and network performance on training, validation and test datasets (accuracy, loss and confusion matrix plots), go to *About* → *About Models* and choose network (Figure 2.3). In current implementation there are two networks available: CNNSimple and VGG19Simple.

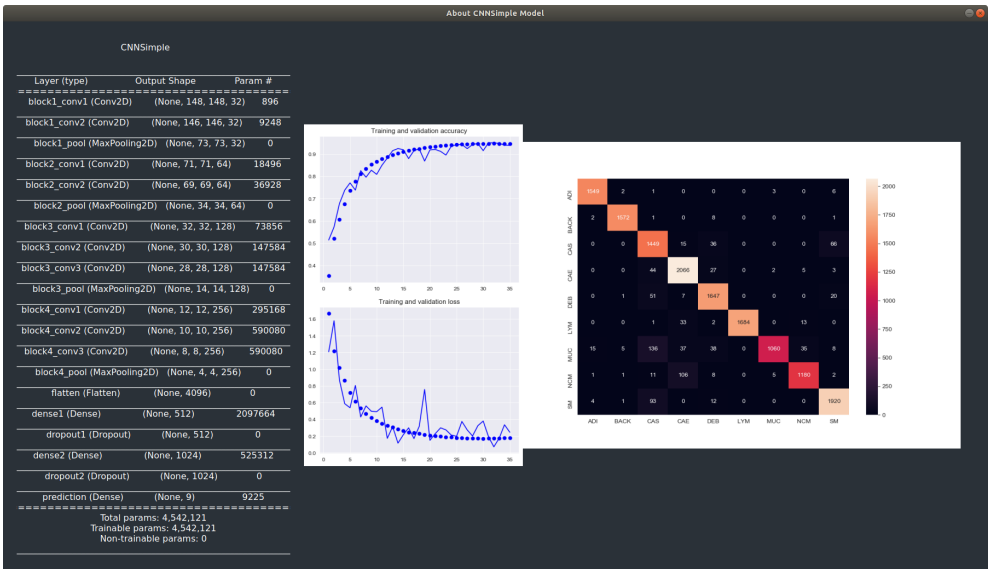


Figure 2.3: CNNSimple neural network basic information

2.5 Basic Use: Predicting Tissue Type

The main use of the Histopathologic Cancer Detection program is to make it easy and fast to load histopathologic slides and get a prediction on tissue and cancer subtype. In order to load a histopathologic slide, click on *Load Image*, and select histopathologic slide for which the prediction is required. Next, select a tissue type of the slide by clicking on *Breast Tissue* or *Colorectal Tissue* radio button. Afterwards, in order to get tissue and cancer subtype, along with probabilities plot, press *Classify* button (Figure 2.4).

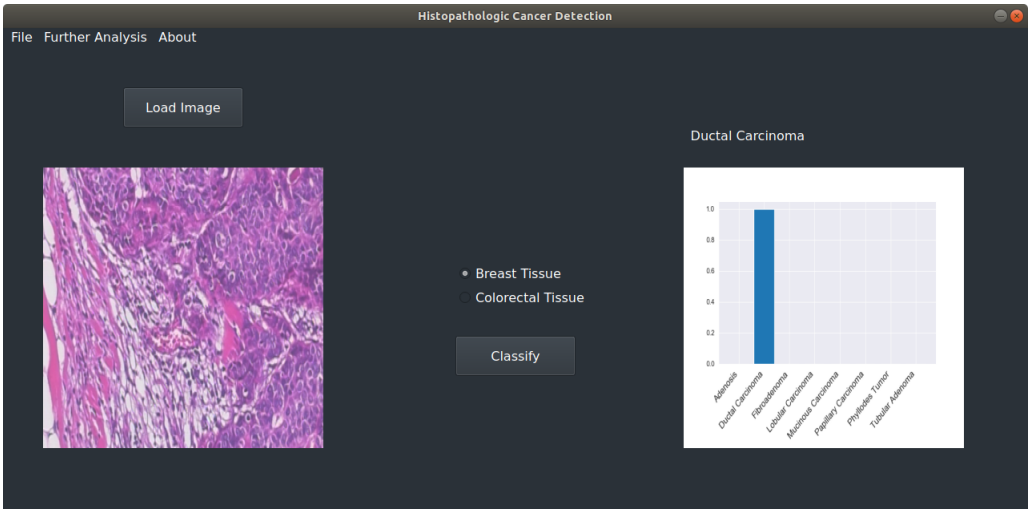


Figure 2.4: Prediction of input histopathologic slide of breast tissue to be ductal carcinoma

2.6 Advanced Use: Visualizing Network Representations

Deep neural networks are highly complex models that have great expressive power and can achieve high accuracy while solving a wide range of problems. Unfortunately, with high complexity comes low interpretability, which presents an issue, especially in deep learning applications to healthcare. Fortunately, there are several methods to inspect convolutional neural networks, and interpret their output. In this paper, I cover three of them: visualizing heatmaps of class activations, visualizing filters of convolutional layers, and visualizing intermediate activations.

2.6.1 Visualizing Heatmaps of Class Activations

Different parts of an image have different weights in networks decision on tissue and cancer subtype classification, and because of that, it is possible to highlight which parts of an image have the highest influence on the network's output. This class of methods is called class activation map visualization, and it produces heatmaps of class activations over input images. A class activation heatmap is a 2D grid of scores associated with a specific output class, computed for every location in any input image, indicating how important each location is with respect to the class under consideration [20]. In order to produce heatmap, go to *Further Analysis* → *Heatmap* (Figure 2.6).

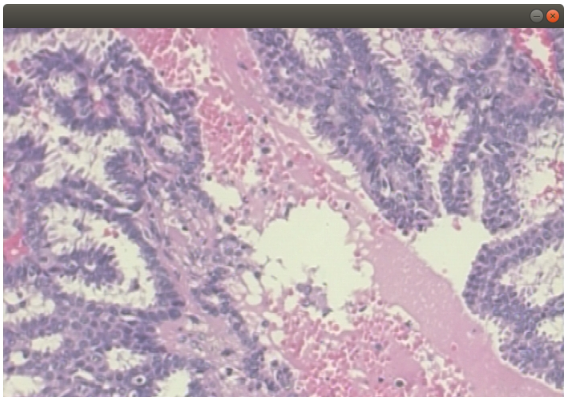


Figure 2.5: Input image of breast tissue (Papillary Carcinoma)

Figure 2.6: Heatmap of class activations on input image

2.6.2 Visualizing Intermediate Activations

Visualizing intermediate activations consists of displaying the feature maps that are output by various convolutional and pooling layers in a network, given a certain input (the output of a layer is often called its activation, the output of the activation function). This gives a view into how an input is decomposed into the different filters learned by the network [20]. In order to visualize intermediate activations of layers, go to *Further Analysis* → *Class Activations*. Next choose which network layer's intermediate activations should be visualized, along with channel number, or 'all', for visualizing all of the layer's channels (Figure 2.7).

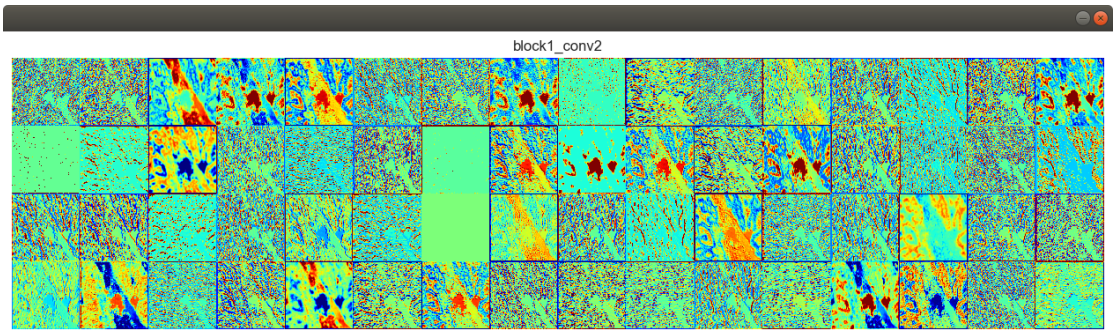


Figure 2.7: Every channel of the second convolutional layer of VGG19Simple on a breast tissue input image (Figure 2.5).

2.6.3 Visualizing Filters of Convolutional Layers

Further analysis is not only dependent on an input image, as we can also inspect feature extractors (filters) of convolutional layers by displaying patterns on which each filter is supposed to respond to. This can be done with gradient ascent in input space: applying gradient descent to the value of the input image of a CNN so as to maximize the response of a specific filter, starting from a blank input image. The resulting input image will be one that the chosen filter is maximally responsive to [20]. In order to visualize filters of convolutional layers, go to *Further Analysis* → *Network Filters*. Next choose which network convolutional layer's filters should be visualized, along with filter number, or 'all', for visualizing all of the layer's filters (Figure 2.8).

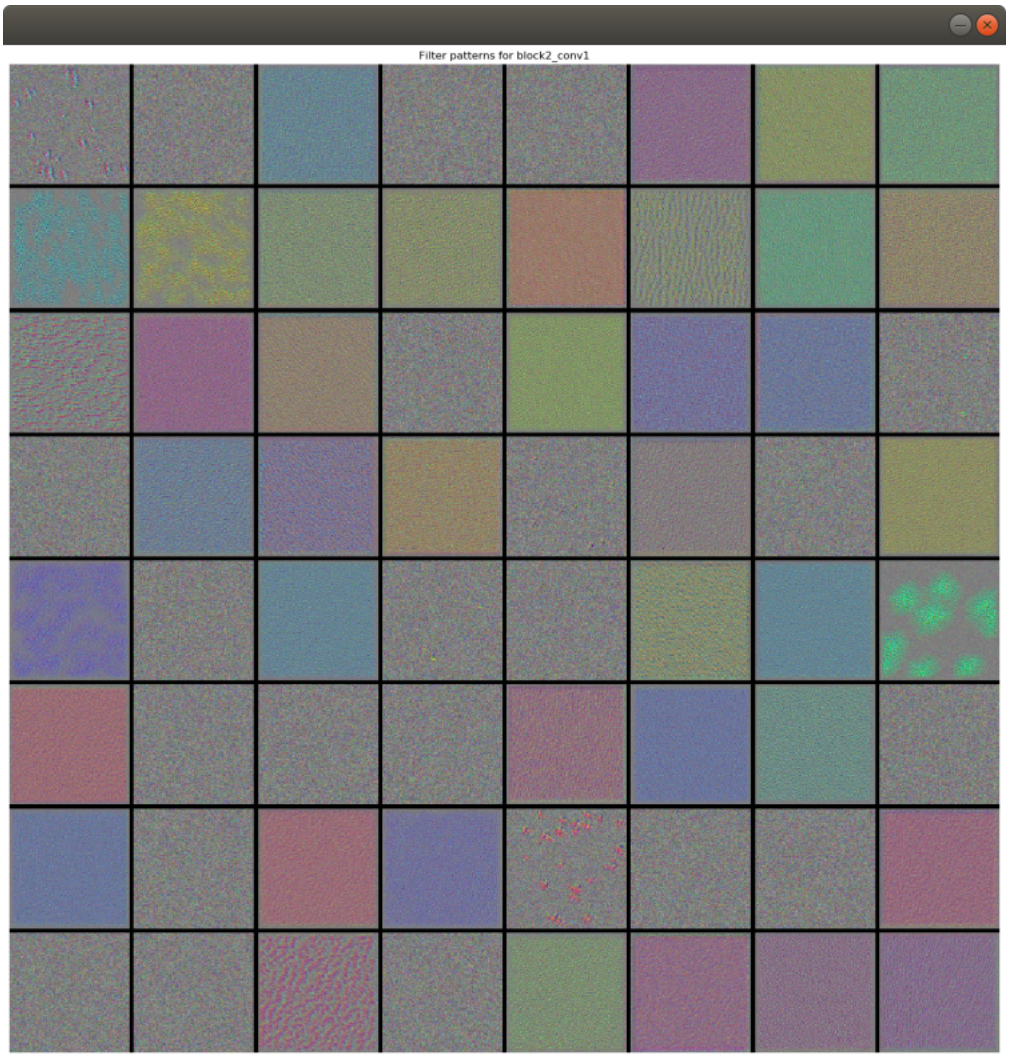


Figure 2.8: Filter patterns for the third convolutional layer of CNNSimple network

2.7 Saving Results

After all of the predictions have been made, and all further analysis has been conducted, in order to save the results, i.e. save all the images created during the program's execution, go to *File* → *Save*.

Chapter 3

Developer Documentation

Creation of the program for histopathologic cancer detection is divided into four major parts:

1. Assembling dataset structure (required for input to a convolutional neural network), image preprocessing (loading, removing noise, normalization, whitening) and data augmentation (expanding the size of the dataset by applying a series of random transformations to each image)
2. Building the convolutional neural networks (class of deep neural networks applied to analyze images), and training them on data
3. Improving prediction accuracy of networks (solving underfitting and overfitting problems) with hyperparameter tuning (choosing optimal parameters for learning algorithm) and changes to network architecture
4. Creating a graphical user interface for the program, which allows user to load histopathologic slide and select network which has to be applied on it, and to get as output category to which that slide belongs, with the additional possibility to visualize network representations (heatmaps of class activations, filters of convolutional layers and intermediate activations)

In the remainder of this chapter each part will be thoroughly analyzed, and directory/file structure of source the code will be illustrated, as well as use-case and class diagrams.

System requirements discussed in user documentation (Section 2.1) apply to developer documentation as well.

3.1 Program Structure

Histopathologic Cancer Detection program is divided into six major modules (Figure 3.1, Figure 3.2):

1. Data - includes dataset creation (Section 3.4)
2. Models - includes creation, training and testing of CNNs (Section 3.5, Section 3.6)
3. Experiments - includes hyperparameter tuning (Section 3.7)
4. Graphical User Interface - includes creation of all application windows and their interconnection (Section 3.8)
5. Utilities - includes dataset analysis and visualization and network performance assessment and visualization (Section 3.9)
6. Tests - includes unit testing of Data, Models, Experiments and Utilities (Section 3.11)

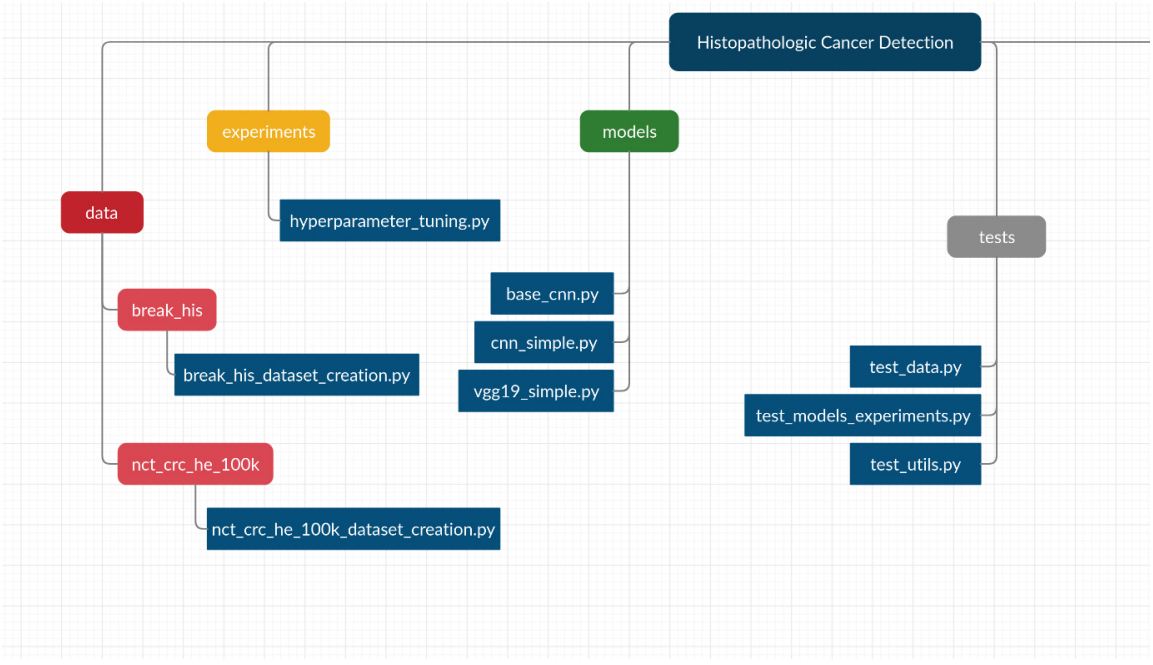


Figure 3.1: Diagram of directories and Python scripts of Data, Models, Experiments and Tests modules of Histopathologic Cancer Detection

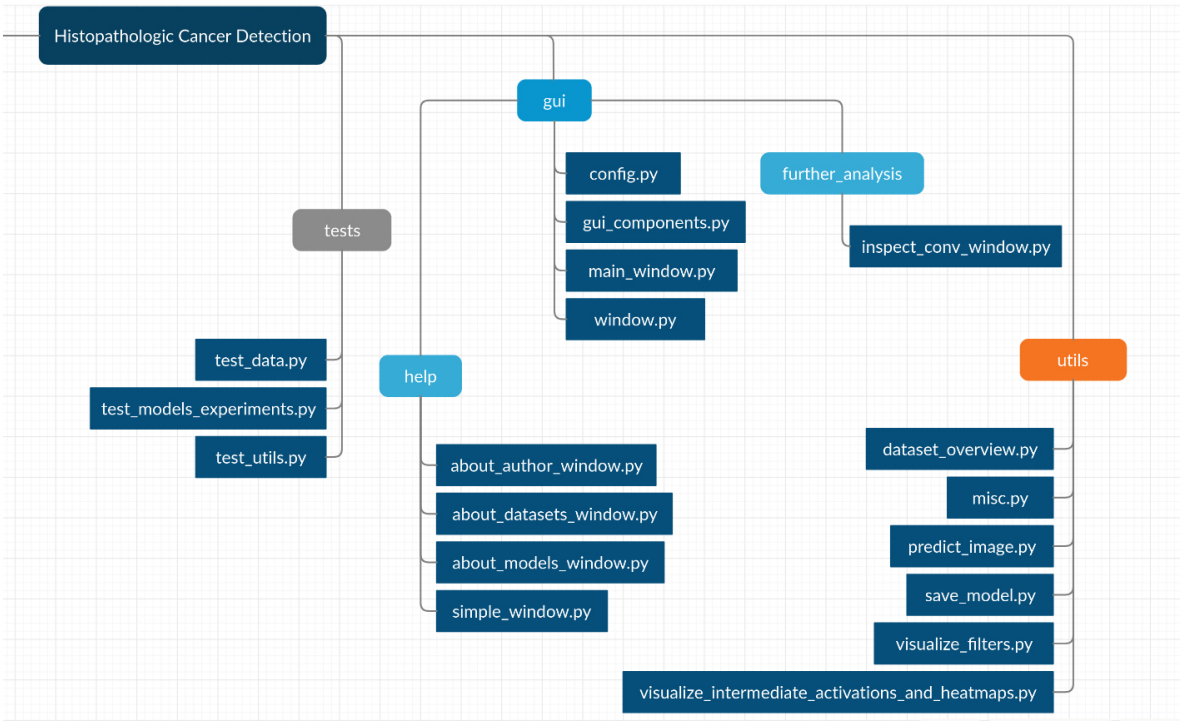


Figure 3.2: Diagram of directories and Python scripts of Tests, Graphical User Interface and Utilities modules of Histopathologic Cancer Detection

3.2 Use-Case Diagram

One of the main goals of the Histopathologic Cancer Detection program was the ease of use, i.e. straightforward graphical user interface which makes complex operations look quite simple and effortless. Even though there are extremely advanced algorithms with millions of parameters behind the program, GUI was made in such a way that everyone can use it. The first step is loading the image and selecting tissue type (breast or colorectal tissue), after which classification is being done. At every step of the way, current work can be saved, and a new image can be loaded to start the process from scratch. After the classification, it is possible to visualize network representations and perform further analysis of the results by visualizing layer activations, network filters, and heatmaps (Figure 3.3).

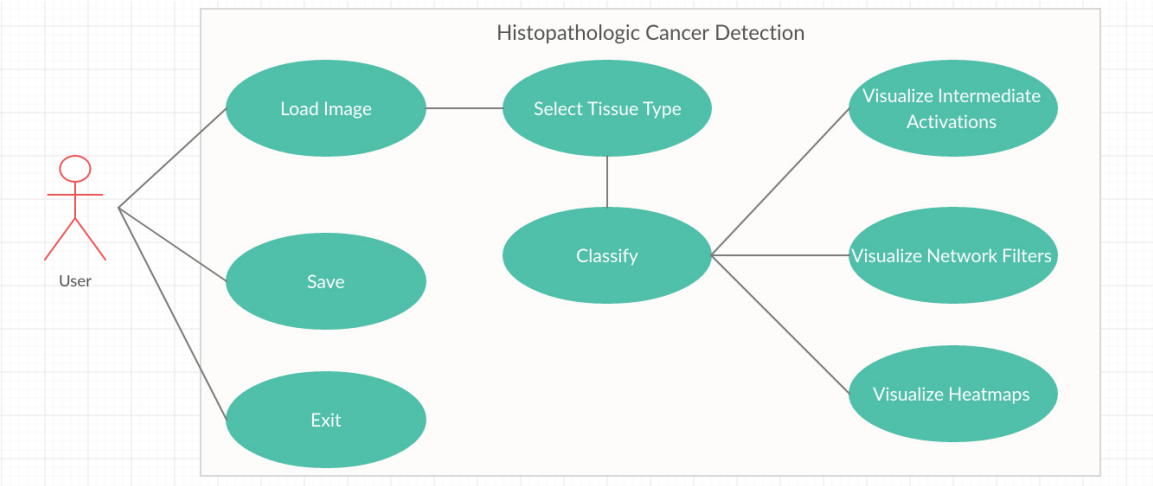


Figure 3.3: Use-Case Diagram of Histopathologic Cancer Detection

3.3 Class Diagrams

Classes of Histopathologic Cancer Detection can be divided into two main segments: window classes and neural network classes.

BaseCNN class is the common class of all neural network classes, and it contains common attributes, such as dataset name, network name, compile parameters, and common methods, such as the creation of data generators, compilation, and training of the network. Neural network classes will be discussed in more detail in Section 3.5, Section 3.6.

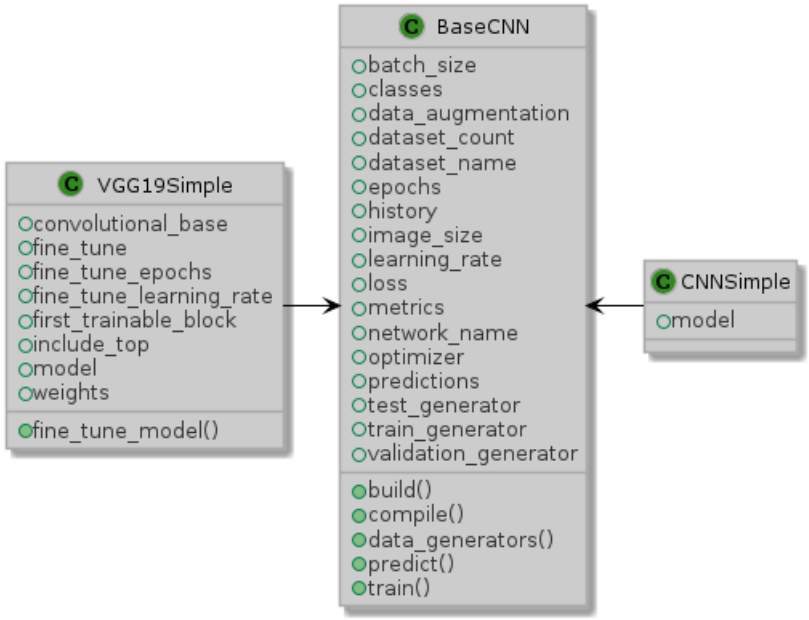


Figure 3.4: Class diagram of neural network classes of Histopathologic Cancer Detection

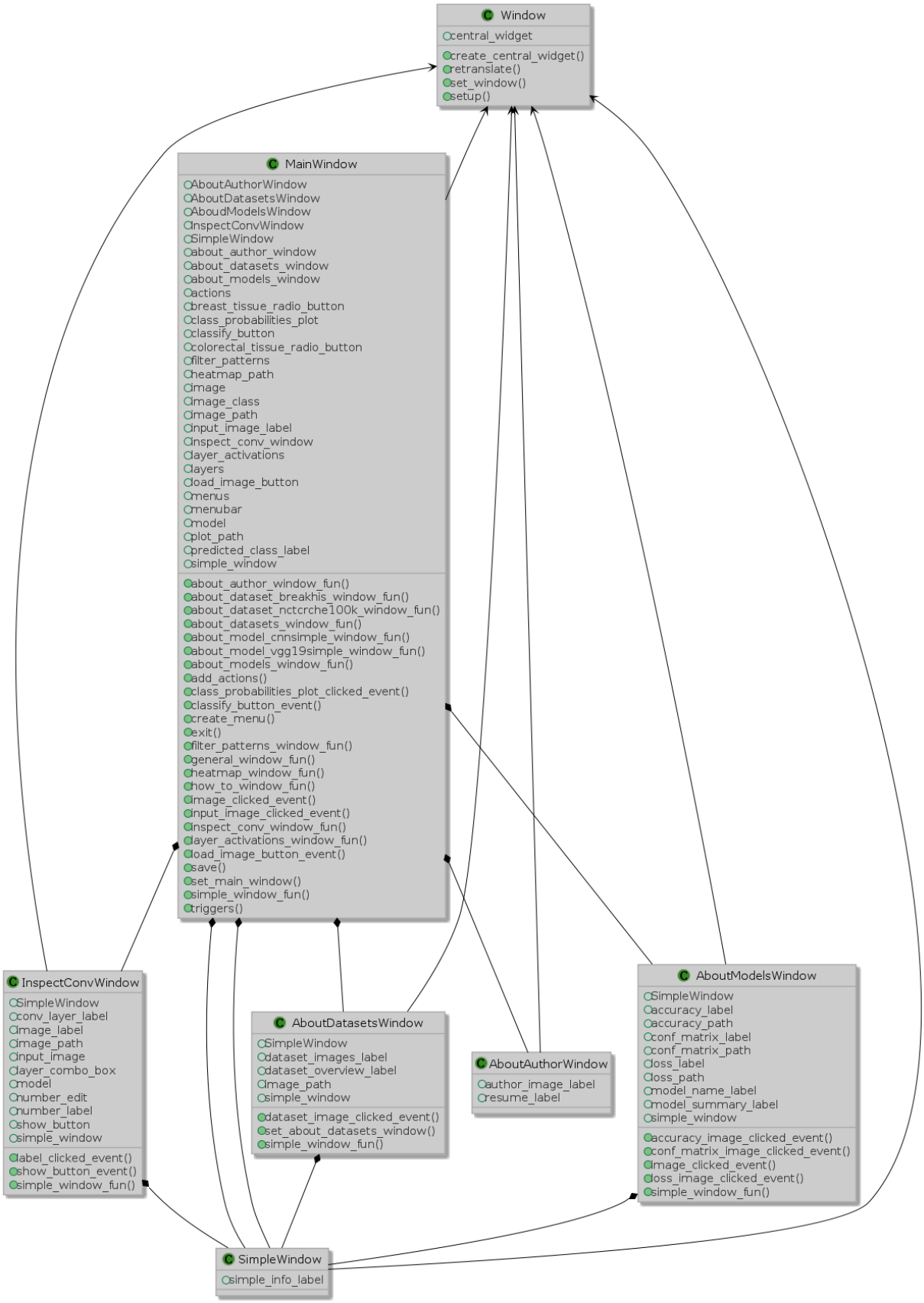


Figure 3.5: Class diagram of window classes of Histopathologic Cancer Detection

Window class is the base class of all window classes, and it implements common methods, such as setting up the window size and central widget. On the other side, MainWindow class is the central point of GUI, as it defines the window which appears when the program is run, and every other window is invoked from it (Figure 3.5). Window classes, along with their attributes and methods, will be discussed in more detail in Section 3.8.

3.4 Creation of Datasets

Performance and accuracy of convolutional neural networks rely largely on datasets, i.e. on quality of available data, dataset size, class balance, etc. But before feeding data to the network, if using `Keras` API, certain dataset structures must be satisfied. More precisely, datasets must have the following structure: train, validation, and test directories, each with a subdirectory for each class. Scripts responsible for the creation of required directory structure and distribution of data are:

- `break_his_dataset_creation.py`,
- `nct_crc_he_100k_dataset_creation.py`.

They work by extracting datasets downloaded from [21], [22], creating necessary directory tree, and distributing images between created subdirectories. After executing scripts, datasets are ready to be fed into convolutional neural networks (in order to train them), but before that, neural network architecture has to be built.

3.5 CNNSimple Implementation

CNNSimple convolutional neural network (Code 3.1) was created using `Keras` Sequential model in order to classify images from NCT-CRC-HE-100K dataset, which contains 100.000 images divided into 9 tissue/cancer categories.

The convolutional base of CNNSimple is composed of four blocks of convolutional and max-pooling layers, where the first two blocks have two convolutional and one max-pooling layer, and the last two blocks have three convolutional and one max-pooling layer. Each convolutional layer has a 3×3 convolution window, uses the ReLU activation function, and the number of feature maps (filters) increases exponentially from 32 to 256. Each max-pooling layer has pool size of 2×2.

Classification top of CNNSimple is composed of two fully-connected layers, each followed by a dropout layer, and an output (also fully-connected) layer. Fully-connected layers use ReLU activation function, and number of neurons grows from 512 to 1024. Dropout layers use 50% dropout rate (fraction of neurons which will be ignored in each passing). Output layer uses softmax activation function, and has nine neurons (one neuron per tissue/cancer subtype output).

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3),
                 name='block1_conv1'))
model.add(Conv2D(32, (3, 3), activation='relu', name='block1_conv2'))
model.add(MaxPooling2D((2, 2), name='block1_pool'))
model.add(Conv2D(64, (3, 3), activation='relu', name='block2_conv1'))
model.add(Conv2D(64, (3, 3), activation='relu', name='block2_conv2'))
model.add(MaxPooling2D((2, 2), name='block2_pool'))
model.add(Conv2D(128, (3, 3), activation='relu', name='block3_conv1'))
model.add(Conv2D(128, (3, 3), activation='relu', name='block3_conv2'))
model.add(Conv2D(128, (3, 3), activation='relu', name='block3_conv3'))
model.add(MaxPooling2D((2, 2), name='block3_pool'))
model.add(Conv2D(256, (3, 3), activation='relu', name='block4_conv1'))
model.add(Conv2D(256, (3, 3), activation='relu', name='block4_conv2'))
model.add(Conv2D(256, (3, 3), activation='relu', name='block4_conv3'))
model.add(MaxPooling2D((2, 2), name='block4_pool'))
model.add(Flatten(name='flatten'))
model.add(Dense(512, activation='relu', name='dense1'))
model.add(Dropout(0.5, name='dropout1'))
model.add(Dense(1024, activation='relu', name='dense2'))
model.add(Dropout(0.5, name='dropout2'))
model.add(Dense(9, activation='softmax', name='prediction'))
```

Code 3.1: CNNSimple network architecture (defined in `cnn_simple.py`)

3.6 Transfer Learning

Although CNNs are a powerful tool for image classification, in order to achieve high accuracy, a large amount of data is required. The problem occurs when only a small dataset is available (as is often in healthcare, ex. BreakHis dataset). In such cases transfer learning can be used: take a model trained on a large dataset and transfer knowledge to a small dataset, i.e. freeze convolutional base, and only train classification top of the network. The main idea is that early layers of convolutional base learn low-level features applicable across all images, such as edges and patterns.

3.6.1 VGG19Simple Implementation

VGG19Simple convolutional neural network (Code 3.2) was created using `Keras` Graphical API in order to classify images from BreakHis dataset, which contains 2.081 images divided into 8 tissue/cancer categories.

Convolutional base of VGG19Simple is VGG19 [23] pre-built network pre-trained on ImageNet [24] dataset (without top classification part), using imagenet weights.

Classification top of VGG19Simple is composed of two fully-connected layers, each followed by a dropout layer, and an output (also fully-connected) layer. Fully-connected layers use the ReLU activation function, and the number of neurons grows from 512 to 1024. Dropout layers use a 50% dropout rate (fraction of neurons which will be ignored in each passing). The output layer uses softmax activation function, and has eight neurons (one neuron per tissue/cancer subtype output).

```
1  input = Input((150, 150, 3))
2  convolutional_base = VGG19(weights='imagenet', include_top=False,
3                             input_tensor=input)
4  for layer in convolutional_base.layers:
5    layer.trainable = False
6  x = Flatten(name='flatten')(convolutional_base.output)
7  x = Dense(512, activation='relu', name='dense_1')(x)
8  x = Dropout(0.5, name='dropout_1')(x)
9  x = Dense(1024, activation='relu', name='dense_2')(x)
10 x = Dropout(0.5, name='dropout_2')(x)
11 x = Dense(8, activation='softmax', name='predictions')(x)
12
13 model = Model(input, x)
```

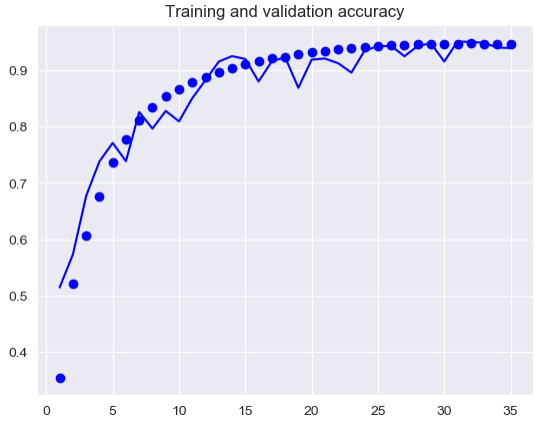
Code 3.2: VGG19Simple network architecture (defined in `vgg19_simple.py`)

3.7 Experiments and Results

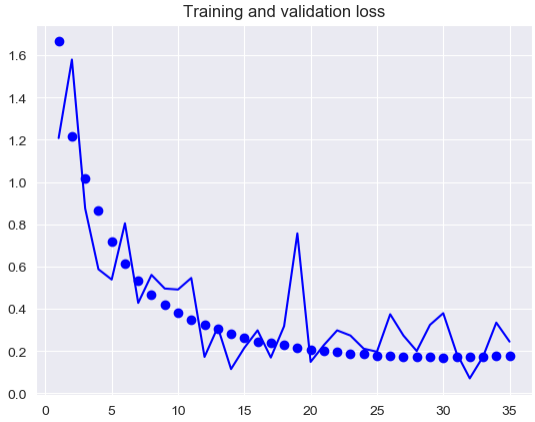
Performance of neural networks is determined by how well will it generalize, i.e. how high accuracy will it achieve on previously unseen data (if it performs well on training data, but underachieves on test data, it is said that CNN overfits). In order to prevent overfitting, a number of techniques can be used: increase the size of the dataset, change network architecture or apply hyperparameter tuning techniques, which consist of selecting a set of optimal hyperparameters for the learning algorithms. The selection of such parameters for networks is done in `hyperparameter_tuning.py`. The first step consists of defining the hyperparameter dictionary with parameters and values to be tested, such as the number of epochs for which the network is to be trained, optimization techniques, etc. The next step consists of training a network with all combinations of parameters and values defined, after which network performances are compared in order to determine the optimal hyperparameter set.

CNNSimple network trained on the NCT-CRC-HE-100K dataset was trained for 35 epochs, using RMSProp optimizer with a learning rate of 0.00004, using a categorical cross-entropy loss function. Before feeding data to the network, data augmentation (applying random transformations in order to produce more images, such as translation, rotation, sheer) has been used. In order to assess the performance of the network, the accuracy metrics function was employed. CNNSimple achieved 93.89% validation and 94.21% test accuracy, and 0.24 validation and 0.06 test loss (Figure 3.6).

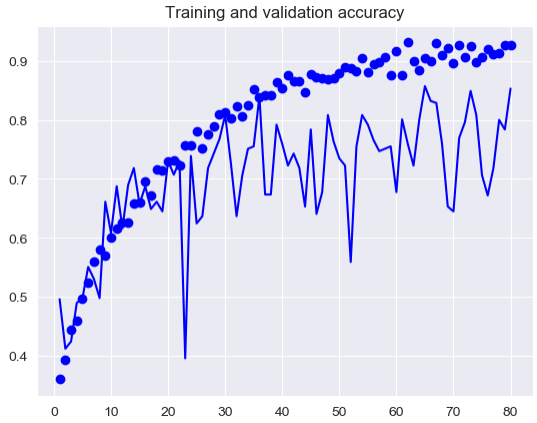
VGG19Simple network trained on the BreakHis dataset was trained for 90 epochs, using RMSProp optimizer with a learning rate of 0.0001, using a categorical cross-entropy loss function. Before feeding data to the network, data augmentation (applying random transformations in order to produce more images, such as translation, rotation, sheer) has been used. In order to assess the performance of the network, the accuracy metrics function was employed. After training a network with a frozen convolutional base, the last two convolutional blocks were unfrozen, and network was trained again using RMSProp optimizer with a learning rate of 0.00004. VGG19Simple achieved 85.3% validation and 83.59% test accuracy, and 0.72 validation and 1.02 test loss (Figure 3.6).



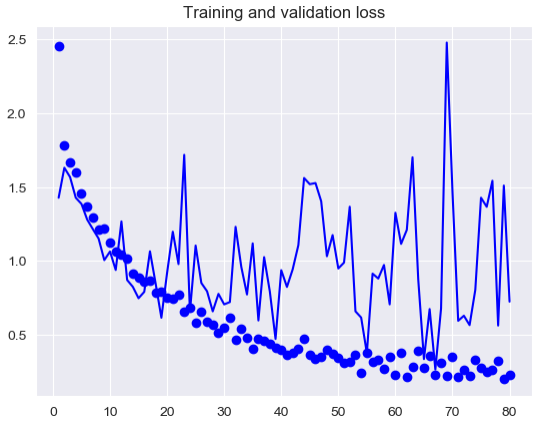
(a) CNNSimple Accuracy



(b) CNNSimple Loss



(c) VGG19Simple Accuracy



(d) VGG19Simple Loss

Figure 3.6: CNNSimple and VGG19Simple performance on BreakHis and NCT-CRC-HE-100K train and validation datasets (respectively)

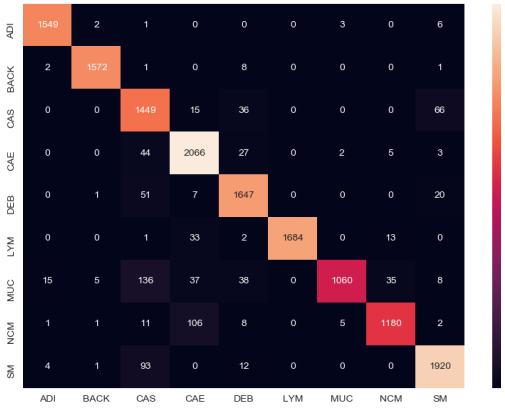


Figure 3.7: Confusion Matrix (a summary table of correct and incorrect predictions broken down by each class) of CNNSimple on NCT-CRC-HE-1OOK test dataset

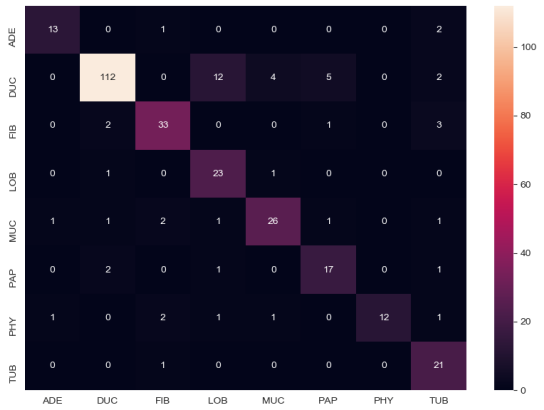


Figure 3.8: Confusion Matrix of VGG19Simple on BreakHis test dataset

3.8 Graphical User Interface

In order to make using program simple and straightforward, graphical user interface using PyQt5 has been created. Following window classes have been constructed:

- Window class in `window.py`, used as base class for every other window, which contains primitive functions for setting up window, creating central widget, retranslating text, etc.
- AboutAuthorWindow class in `about_author_window.py`, which contains two labels (containing image and CV respectively)
- AboutDatasetsWindow class in `about_datasets_window.py`, which contains two labels (containing dataset sample images and dataset overview numbers respectively)
- AboutModelsWindow class in `about_models_window.py`, which contains five labels (containing network name, network architecture, accuracy, loss and confusion matrix plots respectively)
- SimpleWindow class in `simple_window.py`, which contains one image label
- InspectConvWindow class in `inspect_conv_window.py`, which contains three labels (containing convolutional layer text, number text and image respectively), button (with associated action), combo box (containing network layer names) and line edit (for filter/channel selection)
- MainWindow class in `main_window.py`, which connects every other window and provides high-level program functionality

In addition, every window-specific information is located in `config.py`, such as window sizes, positions and names of labels, paths to images, etc. GUI component definitions are located in `gui_components.py`.

3.8.1 Main Window

MainWindow class is the central part of the GUI, as it defines the window which appears when the program is started. It has a menu bar from which every other window can be reached, and a central widget which contains input image, tissue-type radio buttons, classify button and output class label, and probabilities plot.

Function *classifyButtonEvent*, associated with *classifyButton* is an integral part of the class, as it loads network based on the tissue type of the input image, classifies it, and writes output to labels.

3.9 Utilities

For program to work seamlessly, certain utilities needed to be implemented:

- `dataset_overview.py`, used for obtaining basic information about dataset, such as sample images and image distribution per class
- `misc.py`, containing functions for reading file contents, loading images, etc.
- `predict_image.py`, used for predicting class to which an image belongs
- `save_model.py`, used for saving neural network information and performance after being trained, such as arguments, architecture, accuracy, loss and confusion matrix plots, filters, etc.
- `visualize_filters.py`, used for visualizing network filter patterns
- `visualize_intermediate_activations_and_heatmaps.py`, used for visualizing intermediate activations of the network, and heatmaps of class activation

3.10 Implementing Additional Features

In addition to the ease of use of the Histopathologic Cancer Detection program, source code was written in such a way to make expanding scope (problem space) of the program by including additional features quite simple and fast. If new tissue type (ex. lung tissue), along with tissue/cancer subtype classification was to be added to the program, it would be accomplished in the following four steps:

1. **Creating dataset**

   Obtaining dataset of new tissue type, and preparing dataset to be fed to `Keras`-built CNN, which includes creating appropriate directory structure and distributing data

2. **Building CNN**

   Crating new convolutional neural network class inherited from BaseCNN by defining data generator transformations, network architecture, etc.

3. **Fine-tuning CNN**

   Defining hyperparameter dictionary and training neural networks in order to increase classification accuracy and prevent overfitting

4. **Extending GUI**

   Adding an additional radio button to MainWindow class, and extending action associated with classify button, as well as further analysis actions, to use newly created network

3.11 Testing

Source code for Histopathologic Cancer Detection can be divided into two equal chunks: code responsible for CNN-related functionality and code responsible for GUI-related functionality.

The code responsible for CNN-related functionality has been unit-tested using `PyTest` testing framework, and has a code coverage over 93%. Unit tests for data module can be found in `test_data.py`, unit tests for models and experiments modules can be found in `test_models_experiments.py`, and unit tests for utilities can be found in `test_utils.py`.

Code responsible for GUI-related functionality has been tested manually, where numerous histopathologic slides have been loaded and classified, as well as further analyzed, in order to inspect program functionality.

3.12 External Code

Main ideas for advanced use of Histopathologic Cancer Detection, i.e. for further analysis of CNN representations and visualization of heatmaps of class activations, intermediate activations and filters of convolutional layers, as well as some parts of source code, have been taken from [20]. The code is licensed under MIT license.

Chapter 4

Conclusion

Over the past decade convolutional neural networks have achieved results few expected would be possible in such a short time period, and recently they have been successfully implemented in the healthcare industry [25]. Histopathological Cancer Detection program shows that deep learning algorithms can be applied in the field of histopathology, and achieve high performance results in order to reduce workload from doctors, and help speed up the process. More importantly, it overcomes one of the major drawbacks of deep learning algorithms, which is the notion of algorithms being 'black-boxes', i.e. not being highly interpretable. By implementing multiple techniques to further analyze network and its decisions, it is possible to explain and clarify network outputs and determine whether such reasoning is correct or not.

4.1 Future Work

After completing the thesis and creating a program for histopathologic cancer detection on breast and colorectal tissue, there are multiple ways of extending the scope of the program: adding more tissue types in order to increase solving power, adding additional ways to further analyze results, and improving graphical user interface in order to make it even more straightforward to use.

ACDC@LUNGHP[1] challenge provides the dataset for lung cancer histopathologic detection, which could be used for extending the program scope.

[1] Automatic Cancer Detection and Classification in Whole-slide Lung Histopathology 2019

DigestPath2019[2] challenge provides the dataset for colorectal histopathologic cancer detection, which could be used to improve the accuracy of already existing CNNSimple network. Additional histopathology datasets are publicly available, such as the ECDP2020 grand challenge dataset and ANHIR[3] challenge dataset.

Another way to visually interpret and further analyze CNN classification results and increase the interpretability of the networks is by using deconvolutional neural networks (DCNNs) [26].

[2]Digestive-System Pathological Detection and Segmentation Challenge 2019

[3] Automatic Non-rigid Histological Image Registration

Appendix A

Convolutional Neural Networks

Convolutional Neural Networks for image classification [27] take an image as an input, process it, and output category to which that image belongs. The processing part consists of a series of layers through which image is propagated in order to learn features, which in turn determine to which class an image belongs. (Figure A.1).

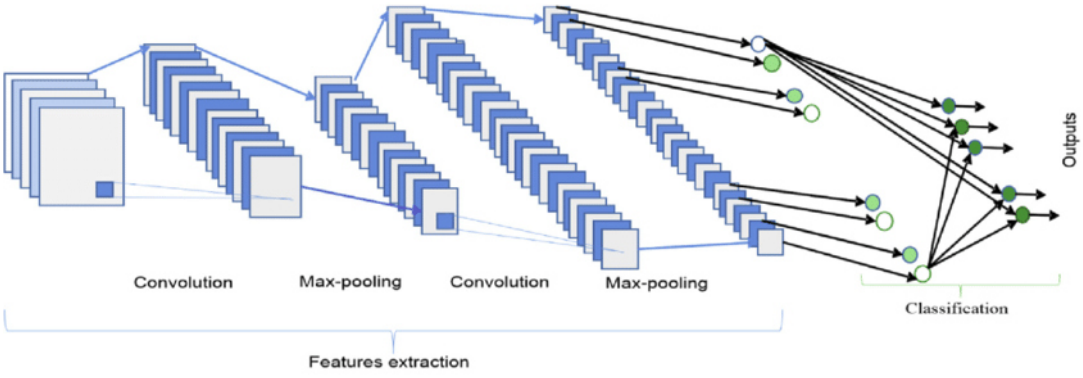


Figure A.1: Convolutional Neural Network consisting of two Convolutional layers, two Max-Pooling layers, Flatten layer and two Fully-Connected (Dense) layers, source [28]

The most commonly used layers in CNN architectures are convolutional layer, max-pooling layer, flatten layer, dense layer, and dropout layer.

A.1 Convolutional Layer

The convolutional layer is the building block of the CNN architecture. Its primary purpose is to extract features from an input image, such as edges, lines, curves, colors. As we go deeper inside the network, it starts identifying more complex features, such

as shapes, objects. This layer consists of multiple filters (feature extractors, usually 3×3 matrices) whose parameters need to be learned.

A.2 Max-Pooling Layer

The max-pooling layer is located after a series of convolutional layers in CNN architecture. It is a downsampling method that reduces dimensionality, thus decreasing the number of parameters and computational power needed in order to train the network, while retaining important features and patterns. It is achieved by applying a max filter to non-overlapping subregions (usually 2×2 matrices), thus reducing the size of each feature map by a factor of 2.

A.3 Flatten Layer

The output of the convolutional base of the network (series of convolutional and max-pooling layers) is a two-dimensional matrix, and before feeding that data to the classification top of the network, it needs to be transformed. Flatten layer reshapes the output matrix to a vector, thus removing all dimensions but one in the process, making the data prepared for the series of fully-connected layers.

A.4 Fully-Connected Layer

After the high-level features of the image have been detected, a series of fully-connected (dense) layers are attached to the top of the network in order to classify an image into a label. Dense layers consist of a huge number of nodes (neurons), which provide a way of learning non-linear combinations of features outputted by a convolutional base, and determine which features most correlate to a particular class.

A.5 Dropout Layer

The fully-connected layer contains the most parameters in the network, and as a result neurons develop co-dependency amongst each other during training,

which leads to overfitting the data (not generalizing well on new, unseen images). In order to prevent that, dropout layers are positioned right after dense layers in CNN architecture as a means of regularizing the network. Dropout consists of randomly ignoring (dropping out) fraction of neurons of fully-connected layer, which in turn makes network learn more robust features, and achieve better performance.

Bibliography

[1] J. Dean, D. Patterson, and C. Young. "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution". In: *IEEE Micro* 38.2 (2018), pp. 21–29.

[2] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[3] Hoang Nguyen et al. "Deep learning methods in transportation domain: a review". In: *IET Intelligent Transport Systems* 12.9 (2018), pp. 998–1004.

[4] EV Polyakov et al. "Investigation and development of the intelligent voice assistant for the Internet of Things using machine learning". In: *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*. IEEE. 2018, pp. 1–5.

[5] Johan Perols. "Financial statement fraud detection: An analysis of statistical and machine learning algorithms". In: *Auditing: A Journal of Practice & Theory* 30.2 (2011), pp. 19–50.

[6] Tom Young et al. "Recent trends in deep learning based natural language processing". In: *ieee Computational intelligenCe magazine* 13.3 (2018), pp. 55–75.

[7] Athanasios Voulodimos et al. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[8] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review". In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[9] Zhong-Qiu Zhao et al. "Object detection with deep learning: A review". In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[10] Richard Zhang, Phillip Isola, and Alexei A Efros. "Colorful image colorization". In: *European conference on computer vision*. Springer. 2016, pp. 649–666.

[11] Ge Wang et al. "Image reconstruction is a new frontier of machine learning". In: *IEEE transactions on medical imaging* 37.6 (2018), pp. 1289–1296.

[12] Chao Dong et al. "Learning a deep convolutional network for image super-resolution". In: *European conference on computer vision*. Springer. 2014, pp. 184–199.

[13] Scott Reed et al. "Generative adversarial text to image synthesis". In: *arXiv preprint arXiv:1605.05396* (2016).

[14] Alvin Rajkomar et al. "Scalable and accurate deep learning with electronic health records". In: *NPJ Digital Medicine* 1.1 (2018), p. 18.

[15] Ryan Poplin et al. "Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning". In: *Nature Biomedical Engineering* 2.3 (2018), p. 158.

[16] Rasool Fakoor et al. "Using deep learning to enhance cancer diagnosis and classification". In: *Proceedings of the international conference on machine learning*. Vol. 28. ACM New York, USA. 2013.

[17] Alexey A Shvets et al. "Automatic instrument segmentation in robot-assisted surgery using deep learning". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2018, pp. 624–628.

[18] Oliveira L. S. Petitjean C. Heutte L. Spanhol F. "A Dataset for Breast Cancer Histopathological Image Classification". In: *IEEE Transactions on Biomedical Engineering (TBME)* 63 (2016), pp. 1455–1462.

[19] Jakob Nikolas Kather, Niels Halama, and Alexander Marx. *100,000 histological images of human colorectal cancer and healthy tissue*. Version v0.1. Apr. 2018. DOI: 10.5281/zenodo.1214456. URL: https://doi.org/10.5281/zenodo.1214456.

[20] Francois Chollet. "Deep learning with Python". In: (2018), pp. 160–177.

[21] *BreakHis Dataset*. Apr. 25, 2020. URL: `https://www.kaggle.com/ambarish/breakhis`.

[22] *NCT-CRC-HE-100K Dataset*. Apr. 25, 2020. URL: `https://zenodo.org/record/1214456#.XpClRPlS8Uo`.

[23] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[24] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[25] Riccardo Miotto et al. "Deep learning for healthcare: review, opportunities and challenges". In: *Briefings in bioinformatics* 19.6 (2018), pp. 1236–1246.

[26] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. "Adaptive deconvolutional networks for mid and high level feature learning". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2018–2025.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[28] Md Zahangir Alom et al. "A state-of-the-art survey on deep learning theory and architectures". In: *Electronics* 8.3 (2019), p. 292.