UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

eXascale Infolab

UNIVERSITY OF FRIBOURG

MASTER THESIS

P-Hydra: Bridging Transfer Learning And Multitask Learning

Author:
Jiyoung Lee

Supervisor:
Prof. Dr. Philippe Cudré-Mauroux

Co-Supervisor:
Dr. Giuseppe Cuccu

December 10, 2020

eXascale Infolab
Department of Informatics

Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland
phone +41 (26) 300 84 65 • diuf-secr@unifr.ch • www3.unifr.ch/inf

Abstract

Jiyoung Lee

P-Hydra: Bridging Transfer Learning And Multitask Learning

Applying deep learning algorithm to cancer detection is an alternative choice for diagnosing cancer. As the importance of detecting cancer in early stage increases, deep learning applications for medical imaging have gained great attention in the research field. Deep convolutional neural networks (CNN), in particular, are widely used in image classification tasks. Due to insufficient amount of medical data, special learning techniques have to be applied. Transfer learning and multitask learning are two popular deep learning algorithms to address this problem. This work focuses on improving cancer detection algorithms by proposing a new learning method called P-Hydra. After introducing fundamentals of deep learning, transfer learning and multitask learning are thoroughly studied as these are essential concepts to understand the proposed method. The suggested model makes use of three different datasets of body parts (prostate, brain, and lung) to make a generalized feature extractor, so that it can be used in multiple medical imaging applications. In addition to the generalized feature extractor, a decision maker for each dataset is created. This experiment showed that training the individual tasks together in parallel improve the performance of a model. The AUC of each model (prostate, brain, and lung) after generalization are 62, 87, and 60 respectively. After specialization, the AUC achieved on the test set are 54, 84, and 60.

Keywords: Machine learning (ML), Convolutional Neural Network (CNN), Multitask Learning, Machine Learning (ML), Cancer Detection, Deep Learning (DL)

Contents

Abstract **iii**

1 Introduction **1**
1.1 Overview . . . 1
1.2 Contributions . . . 2

2 Fundamentals **3**
2.1 Deep Learning . . . 3
2.1.1 Neural Networks . . . 3
2.1.2 Activation Function . . . 4
2.1.3 Cost Function . . . 5
2.1.4 Optimization . . . 6
2.1.5 Backpropagation . . . 7
2.1.6 Regularization . . . 9
2.1.7 Confusion Matrix . . . 10
2.1.8 Evaluation Metrics . . . 11
2.2 Convolutional Neural Networks . . . 12
2.3 Medical Data . . . 13
2.3.1 Cancer . . . 13
2.3.2 Data Types . . . 14
2.3.3 DICOM . . . 15
2.3.4 Manipulating DICOM images . . . 15
2.4 Dataset Description . . . 17
2.5 Related Work . . . 20

3 Transfer Learning **23**
3.1 Transfer Learning . . . 23
3.2 Transfer Learning Application Review . . . 24
3.3 Process Overview . . . 24
3.3.1 Evaluation metric . . . 25
3.3.2 Dataset 1 (Prostate) . . . 25
3.3.3 Dataset 2 (Brain) . . . 25
3.3.4 Dataset 3 (Lung) . . . 25
3.3.5 Dataset 4 (Prostate) . . . 25

4 Multitask Learning **27**
4.1 Multitask Learning and Single task Learning . . . 27
4.2 How does MTL work? . . . 28
4.3 Related tasks . . . 28
4.4 MTL Mechanism . . . 29
4.5 Parallel and Sequential Transfer . . . 29
4.5.1 Parallel Transfer . . . 29
4.5.2 Sequential Transfer . . . 30

5 Proposed Approach **31**
5.1 Different Aspects From Multitask Learning . . . 31
5.2 Goal . . . 31
5.3 Model Architecture . . . 32
5.4 Roulette . . . 32
5.5 Process Overview . . . 34
5.6 Training . . . 35
5.6.1 Evaluation Metric . . . 35
5.6.2 Visualization of a Model . . . 35
5.6.3 P-Hydra Learning Net . . . 35

6 Experimental Results **39**
6.1 Experiments . . . 39
6.1.1 Experimental Setup . . . 39
6.2 Results . . . 39
6.2.1 Discussion . . . 40
DSP . . . 40
DSB . . . 41
DSL . . . 41

7 Conclusion **45**
7.1 Future Work . . . 46

A Appendix **47**
A.1 Script Options . . . 47

Bibliography **49**

List of Figures

2.1 A simple neural network. The graph model on the left represents 3 input neurons connected to each activation function neuron. A non-linear computation of weighted sum appears on the right side. 4
2.2 Sigmoid activation function. The graph's output is between 0 and 1. When the input is too small or too big, the derivative becomes 0 since the slope is flat. 5
2.3 ReLU activation function. The negative input value will always be zero as shown in the graph. The positive region follows a $f(x) = x$ graph. 6
2.4 ELU activation function. It has a negative output when the input is less than 0. It still has a flat region when the input is too small. When the input is positive, the output value is equal to the input. 7
2.5 Gradient descent. The leftmost dot represents initial weights. As training proceeds, the weights take steps towards the bottom. The red dot in the bottom is the point where the weights is at its minimum cost. 7
2.6 Dropout network. When applying dropout, neurons are randomly deactivated (dropped) by dropout probability. Neurons that are removed from the network are marked with red X. Dropout improves the network's performance by reducing overfitting. 10
2.7 Confusion matrix of a binary classification problem. It is a 2 x 2 matrix that shows 4 possible results as well as t he 4 combinations with its corresponding class (positive or negative). 10
2.8 A CNN architecture example. Input goes through three Conv Layers (convolutional layer) and afterwards to 2 FC Layers (fully connected layer). Between each Conv Layer and after the last Conv Layer (Conv Layer 3), batch normalization (batch norm), ELU activation function and dropout are applied. 13
2.9 A basic convolution calculation. The 5 x 5 input matrix is calculated with a 3 x 3 convolution filter. The example of the convolution calculation is presented above. The result of the convolution calculation is 3 x 3 output matrix which is shown on the right. 14
2.10 Different types of padding and consequent output size. Each convolution process is shown. In the case of "valid" no padding is applied, making the output smaller than the input. For "same", padding of 1 is applied and the input is surrounded by one extra pixel. The output size is the same as the input size. For "full" type, padding of 2 is added to the input. After convolution, output size becomes larger than the input size. 15
2.11 Max pooling with 2 x 2 filter, stride of 2. The left matrix represents the input before the max pooling. The right matrix shows the result of the max pooling. 16

2.12 Different sequences of a prostate MRI image. Tumors have white shade in T2-Weighted images and ADC images. In DWI images, tumors have black shade. 16
2.13 Example of the DSP. An MRI image with different sequences, T2-weighted (T2), DWI, and ADC, are presented. In T2 and DWI images, cancerous nodules are shown as white, whereas they appear as black in an ADC image. 18
2.14 Example of DSB. The left picture shows a brain MRI image with a malignant tumor in it. The tumor is shown as white shade on the left side of the image. The right picture is a brain MRI image that does not contain a cancerous tumor. 19
2.15 Example of DSL. The image shows a lung CT scan image that has a cancerous tumor. Usually, a normal brain appears as black. Nodules are shown as white spots. 20

3.1 Transfer learning process. The left column shows a traditional machine learning where task A is trained with the model A. The right column represents a transfer learning where the knowledge gained from training task A is passed on to the model B. 23

4.1 Traditional single task learning. Three machine learning models are presented. Each task is considered as an independent model, thus trained separately. They do not share any layers. [28] 27
4.2 Example of multitask learning. The task 1, 2, and 3 are different tasks but they are trained at the same time. All the tasks are trained on the same input. Three tasks share hidden layers. [28] 28

5.1 Model architecture. The left column indicates a model architecture with a description of each layer. The model is divided into two parts: feature extractor and decision maker. The right column shows a layer structure of a convolutional block and a fully connected block. 33
5.2 Process overview. Each model for each dataset is initialized from the roulette result. Feature extractor (FE) and decision maker (DM) for each dataset is combined as a whole model. Each model is trained for 200 epochs. After 200 epochs, each model is tested on the test set. . . . 36
5.3 End-To-End model in training. For each model, End-To-End model consists of a shared feature extractor and a respective decision maker. . 37
5.4 P-Hydra net. Input is a medical image. It goes through the feature extractor (FE) and useful features for every dataset are extracted. Then each decision maker (DM) is trained individually for the particular dataset. The FE is shared across models, while the DMs are distinct. . . 37

6.1 Final result. DSP with learning rate 1e-5, dropout of 0.3. Accuracy, AUC, f1score, loss, precision, recall, and specificity graphs are illustrated. The number of epochs are shown on Y-axis. X-axis represents a training and validation metric. 42
6.2 Final result. DSB with learning rate 1e-5, dropout of 0.2 43
6.3 Final result. DSL with learning rate 1e-5, dropout of 0.3 44

Chapter 1

Introduction

1.1 Overview

Diagnosing cancer in an early phase is one of the most important factors to increase the survival rate of patients. According to the World Health Organization (WHO) [1], "when cancer care is delayed or inaccessible, there is a lower chance of survival, greater problems associated with treatment, and higher costs of care. Early diagnosis improves cancer outcomes by providing care at the earliest possible stage and is therefore an important public health strategy in all settings."

Skilled radiologists are needed to correctly diagnose cancer. However, the lack of radiologists is a serious problem in many countries. The Clinical Radiology UK Workforce Census Report in 2018 [2] shows that the number of radiologists from the UK's National Health Service cannot adequately respond to the high demand of their services, which already resulted in delayed cancer diagnoses and inadequate emergency diagnostic as well as interventional services. [3] Several strategies have been discussed to overcome this issue. One of the most prominent ones is a diagnosis system based on artificial intelligence.

This paper focuses on the detection of prostate, lung, and brain cancer using deep neural networks and the importance of the medical imaging application of deep learning. It introduces a new algorithm called "P-Hydra" for cancer detection which could improve transfer learning and multitask learning.

There are various types of a cancer screening test. For prostate cancer, Digital rectal examination (DRE), Prostate-specific antigen (PSA) test and Magnetic resonance imaging (MRI) are frequently used. For lung cancer, Low-dose helical or spiral computed tomography (CT or CAT) scan is most commonly used. As for brain cancer, MRI is used to detect whether there is a tumor or not. In a recent study by Wallis et al [4], it is shown that MRI-based testing has better performance than PSA alone. This also proves that Computer-Aided Diagnosis using medical images like MRI can play a major role in detecting cancer. Computer-Aided Diagnosis systems, as the name suggests, work as an assistant for radiologists.

With the advent of deep learning and its applications in medical imaging, better algorithms for cancer detection are being studied. As Yoo et al [5] stated, "In medical imaging field, computer-aided detection and diagnosis (CAD), which is a combination of imaging feature engineering and machine learning classification, has shown potential in assisting radiologists for accurate diagnosis, decreasing the diagnosis time and the cost of diagnosis".

Chapter 2 explains all the basics to understand this paper. The concept of deep learning and neural network, including the detailed explanations, are discussed. Medical information concerning cancer and its data format are presented as well.

Thereafter, datasets that are used in this work are thoroughly described with its preprocessing step. Lastly, related work regarding cancer detection that uses a CNN model is introduced.

In Chapter 3, one of the deep learning models, transfer learning, is discussed. Particular focus is given to the new transfer learning technique by Clement et al [6].

Multitask learning is then presented and covered more thoroughly by looking at how it occurs as well as its mechanisms. It is important to understand what the aspects these two learning methods share and how it differs from each other.

A new algorithm , P-Hydra, that aims to improve cancer detection is introduced in Chapter 5. Both the similarities and differences between our approach and transfer learning and multitask learning are discussed.

Lastly, we check the algorithm's validity by applying it on a medical dataset and by reviewing the experimental results of the model.

1.2 Contributions

- A novel algorithm crossing transfer learning and multitask learning.
- A feature extractor that generalizes across different medical image applications.
- State-of-the-art results on three medical applications for autonomous cancer detection.

Chapter 2

Fundamentals

This chapter gives an overview of the essential concepts of deep learning which will help the reader understand this work. Medical data types, dataset description are presented as well with the pre-processing step. It begins by explaining what a neural network is. Then, the essential parts of deep learning, such as cost function, optimization, and regularization are discussed. Lastly, related work of medical imaging application using CNN (Convolutional neural networks) on prostate, brain, and lung are presented.

2.1 Deep Learning

Deep learning is nowadays one of the most popular research topics in machine learning. It refers to training very complex neural network models. Various deep learning applications are used in numerous fields. There are many types of deep learning networks such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short Term Memory Network (LSTM). In this work, we choose CNN because it is suitable for image processing.

2.1.1 Neural Networks

Neural networks originated from the idea of how human brain works. Neurons in the brain are connected by synapses. When they get a stimulus from outside, the stimulus is passed through the axon in the form of electric signals. With this process, neurons pass information to other neurons that are connected to one another. In machine learning, a neuron, or a perceptron is a function that takes in a weighted sum of the outputs of neurons in the previous layer and gives the results of an activation function.

$$o_i = \sigma(\Sigma(Wa_i + b)) \tag{2.1}$$

Figure 2.1 illustrates a simple neural network with a sigmoid activation function σ. Inputs a in the first layer (input layer) are computed with their corresponding weights w and biases b. Depending on the activation function, the result is in a certain range (i.e. between 0 and 1 in case of a sigmoid activation function). In the output (last) layer, we get the result of an activation function of as output o. Equation 2.1 shows the non-linear function of how each output of the simple neural network (Figure 2.1) is calculated. If it is a classification problem, we get the prediction as a class. Training the neural network means finding out the optimal weights and biases for the network.

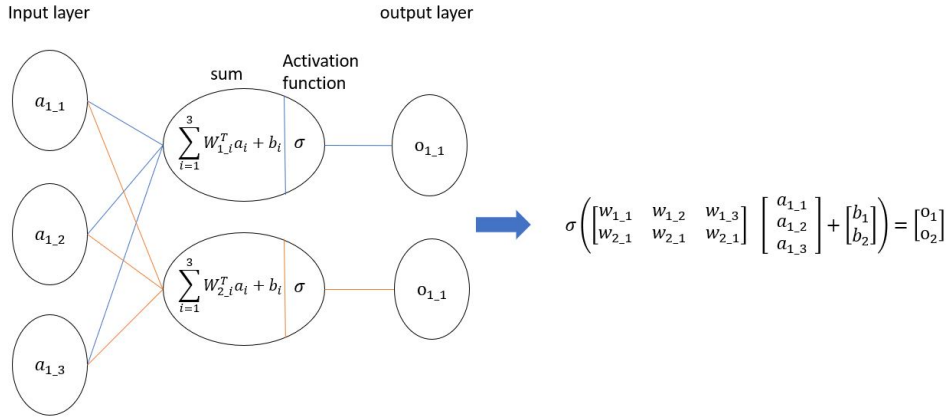

FIGURE 2.1: A simple neural network. The graph model on the left represents 3 input neurons connected to each activation function neuron. A non-linear computation of weighted sum appears on the right side.

2.1.2 Activation Function

After the weighted sum of all inputs (see equation 2.1) from each neuron of the previous layer is done, the result is calculated by a non-linear function which is called an activation function. Nwankpa et al [7] claim that activation functions are needed to convert the linear input signals and models into non-linear output signals. This assists the learning of high order polynomials greater than one degree for deeper networks. In figure 3.1, *sigma* is the activation function. Here are three frequently used activation functions.

- **Sigmoid activation function**

  The formula of the sigmoid activation function is:

  $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

  The output result of a sigmoid activation function is in a range between 0 and 1. Although it is a very popular activation function, it has several drawbacks. First, when you have inputs that are too large or too small, the gradients will be zero, since these two regions have a constant value, making the slope flat. The zero gradient causes a saturation problem. If the gradient is zero, the network becomes hard to learn and performs poorly. Secondly, because it is not zero centered, the gradient can only be updated in one direction, which results in poor performance.

- **ReLU activation function**

  The formula of the ReLU (Rectified Linear Unit) activation function is:

  $$ReLU(x) = max(0, x) \tag{2.3}$$

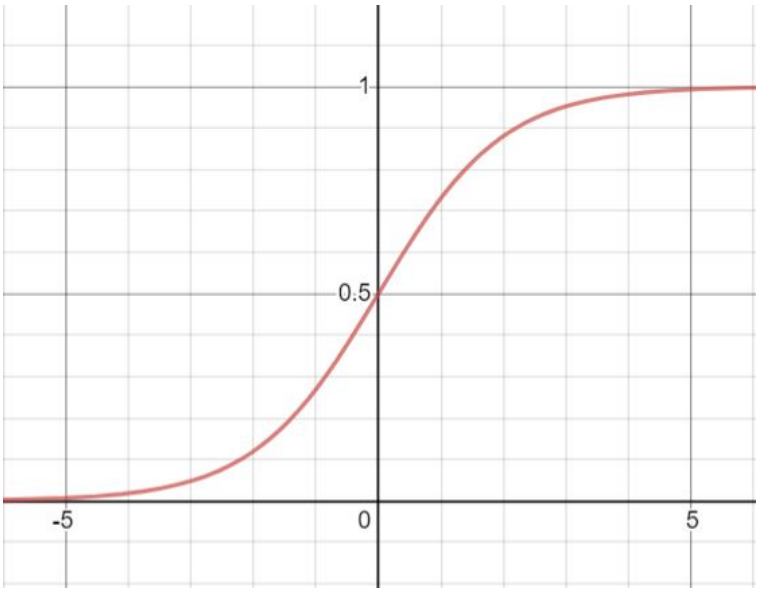

FIGURE 2.2: Sigmoid activation function. The graph's output is between 0 and 1. When the input is too small or too big, the derivative becomes 0 since the slope is flat.

A ReLU Activation function is very popular, especially when it comes to deep learning models. It is also not zero centered, however, unlike the sigmoid activation function, it does not have a saturation problem in the positive region, since the slope is not flat. It is also faster to compute than the sigmoid function since its form is much simpler. The disadvantage of ReLU is that, for the negative inputs, the gradient will still always be zero. Thus, it is unlikely for ReLU to be activated from that state, which means it does not update since the gradient is stuck in the flat region.

- **ELU activation function**

  The formula of ELU (Exponential Linear Unit) activation function is:

$$ELU(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \tag{2.4}$$

  An ELU activation function is similar to the ReLU activation function as shown in figure 3.3, and figure 3.4, except for the inputs that are $x < 0$. The negative region still has the saturation problem, however, its output is closer to zero-mean (mean of the activation output is zero). According to experiments by Clevert et al [8], the ELU activation function could generalize the model better than the ReLU activation function with accelerated learning speed. When having very small gradients (in the negative region), less information is passed to the next layer. The work argues that this makes the ELU activation function more robust to noise. In this work, we use this activation function.

2.1.3 Cost Function

After we get the predictions, the neural network needs to be evaluated by comparing the real labels and the predictions. To construct the error gradient, a cost function

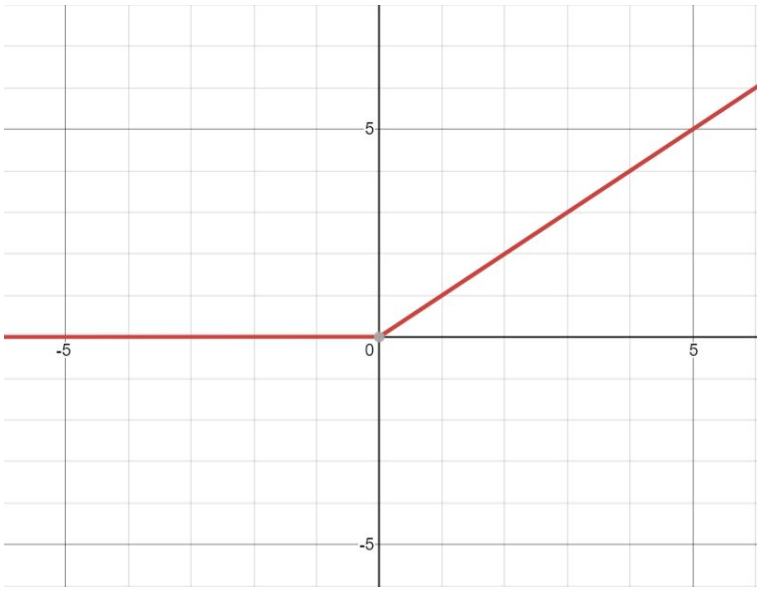

FIGURE 2.3: ReLU activation function. The negative input value will always be zero as shown in the graph. The positive region follows a $f(x) = x$ graph.

is needed. It measures the error (loss) between the actual value and the predicted value. The cost function averages the sum of losses from the loss function. Cross entropy loss function is usually used for classification problems. There are two types of cross entropy loss function: the binary cross entropy loss and the categorical cross entropy loss. The binary cross entropy loss is used when the number of classes is 2. The latter is used when the number of classes more than 2. The binary cross entropy loss function is:

$$Loss_p(y) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{2.5}$$

N is the number of inputs. y refers to an actual label and $p(y)$ refers to a predicted value. The equation computes the difference between the actual value and the predicted one. The categorical cross entropy loss for the multi class classification is:

$$Loss_p(y) = -\sum_{i=1}^{N_c}\sum_{j=1}^{N}(y_{ij} \cdot log(p(y_{ij}))) \tag{2.6}$$

N_C means the number of classes. As mentioned above, learning refers to finding out the optimal weights and biases. The goal of learning is thus to find the optimal parameters that minimize the cost function.

2.1.4 Optimization

How can we minimize the losses of a cost function? It is done by using optimization algorithms. Optimization makes it possible for the model to reduce the losses in an efficient way. Depending on optimizers, different ways to change weights or learning rates are applied. Gradient descent is one of the optimization algorithms that is useful to get a local minimum of a function. Let us assume that you are in the middle of a mountain and you are trying to reach the lowest part of the mountain. You

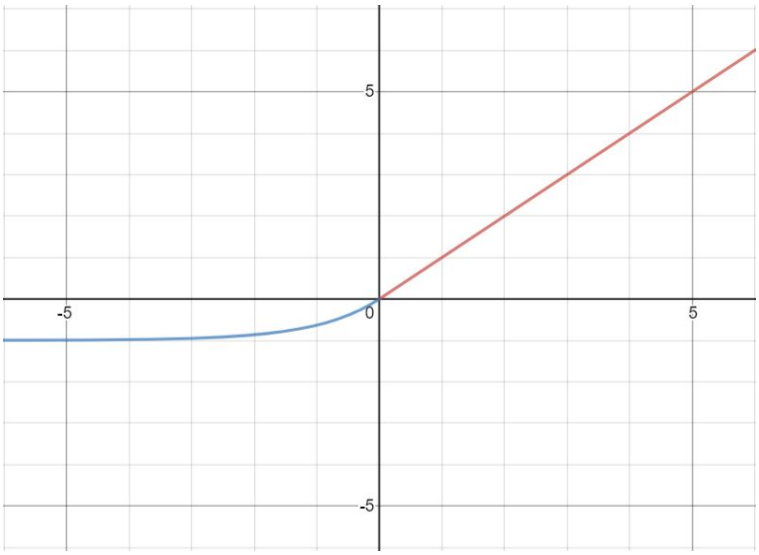

FIGURE 2.4: ELU activation function. It has a negative output when the input is less than 0. It still has a flat region when the input is too small. When the input is positive, the output value is equal to the input.

can either go uphill or downhill and you can choose how much you want to walk per one step. To decide these things, gradient descent is needed. We take downhill steps on the error (cost) slope, which we can know by the negative gradient. Then, we repeat the procedure until the weights reach the lowest point of the graph.

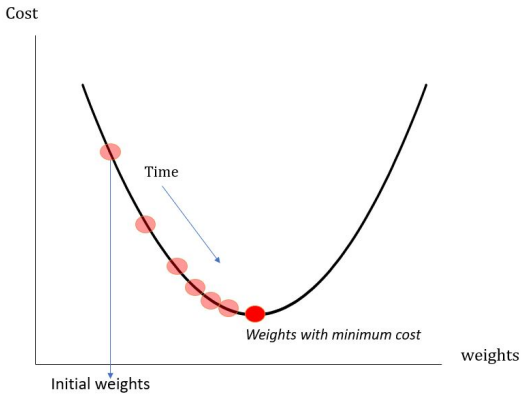

FIGURE 2.5: Gradient descent. The leftmost dot represents initial weights. As training proceeds, the weights take steps towards the bottom. The red dot in the bottom is the point where the weights is at its minimum cost.

2.1.5 Backpropagation

Backpropagation is an algorithm to efficiently minimize the cost of the weights in a neural network. In order to correctly classify the label, we want the probability of the target class to increase. When the output ranges between 0 and 1, an index of the maximum value is selected as a predicted label. As you can see in figure 3.1, the input a which contains initial information propagate through the hidden neurons at

each layer and finally produce the predicted label. This is called forward propagation. Through this feedforward propagation process, the cost is generated. [9] We use the following notation to compute the gradient.

- The error: $E(w) = \Sigma L(f_w(x), y)$ for point (x,y)
- Loss function: L
- Activation function: σ
- The error gradient: $\nabla_w E(w)$
- The expanded error: $E(w) = \Sigma L(\sigma(w^{(l)} \cdot \sigma(w^{(l-1)} \cdot \sigma(...W^{(1)} \cdot x...))), y)$
- The weight matrix for layer k: $w^{(k)}$
- Gradient descent for layer k: $\delta(k) = \frac{\partial E}{\partial u^{(k)}}$
- $E = L(v^{(l)}, y)$ and $v^(l) = \sigma(u^{(l)})$ for k = l (i.e last layer), u is the last layers

First, it starts with calculating the error by summing all the losses. Then, the gradient of the error function is obtainable with the help of the chain rule:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{2.7}$$

The chain rule enables a very complex expression of gradients to be calculated in simpler way. From this process, the backpropagation algorithm is yielded. Let us assume that the loss function is squared loss

$$L(v^{(l)}, y) = \frac{||v^{(l)} - y||^2}{2} \tag{2.8}$$

then, we can obtain the derivative

$$\delta(l) = \frac{\partial L(\sigma(u^{(l)}, y)}{\partial u^{(l)}} = \frac{\partial L(v^{(l)}, y)}{\partial v^{(l)}} \cdot \frac{\partial v^{(l)}}{\partial u^{(l)}} = (v^{(l)} - y) \cdot (1 - \sigma(u^{(l)}) \cdot \sigma(u^{(l)}) \tag{2.9}$$

We need the target class y, the network output $v^{(l)}$, and the sum of the weighted input $u^{(l)}$ to compute equation 2.2. Any loss function L and activation function σ can be applied to this computation. What we are interested in is the gradient of the error

$$\frac{\partial E}{\partial W_{i,j}^{(l)}} \tag{2.10}$$

Let us assume that k = l. The gradient is

$$\frac{\partial E}{\partial W_{i,j}^{(l)}} = \frac{\partial E}{\partial u_i(l)} \cdot \frac{\partial u_i(l)}{\partial W_{i,j}(l)} = \delta_i^{(l)} \cdot v_j^{(l-1)} \tag{2.11}$$

From equation 2.3, $v_j^{(l-1)}$ is induced. Now, let us look at the layers k < l, then the gradient is

$$\frac{\partial E}{\partial W_{i,j}^{(k)}} \tag{2.12}$$

Then, we can apply the same principle.

$$\frac{\partial E}{\partial W_{i,j}^{(k)}} = \frac{\partial E}{\partial u_i(k)} \cdot \frac{\partial u_i(k)}{\partial W_{i,j}(k)} = \delta_i^{(k)} \cdot v_j^{(k-1)} \tag{2.13}$$

Equation 2.4 needs the layer inputs $v^{(k-1)}$ and the backpropagated error $\delta^{(k)}$ We can get the error $\delta^{(k)}$ through the error $\delta^{(k+1)}$

$$\delta^{(k)} = \frac{\partial E}{\partial u^{(k)}} = \frac{\partial E}{\partial u^{(k+1)}} \cdot \frac{\partial u^{(k+1)}}{\partial v^{(k)}} \cdot \frac{\partial v^{(k)}}{\partial u^{(k)}} = \delta^{(k+1)} \cdot W^{(k+1)} \cdot \sigma'(u^{(k)}) \tag{2.14}$$

We get the 'k'th error through 'k+1'th error, so the computation flows from end to front of the network. That is why this algorithm is called backpropagation.

2.1.6 Regularization

Even if the model performs well on the training data, we want a model that is generalized, which is the model that can predict well even on the unseen data. In other words, we are trying to avoid the overfitting problem. Overfitting happens when the model remembers every detail and noise of the certain (training) data, thus fails to make a good prediction on the new data. There are several regularization methods to avoid this problem, such as data augmentation, early stopping, and dropout.

- **Data Augmentation**

  One of the ways to fix overfitting is to train on more data. Realistically speaking, obtaining more data might not be possible in some cases. However, with the help of data augmentation, we can generate artificial data. Artificially generated data can be added to a training set, helping the model improve its performance. Datasets can be augmented by applying transformation such as flipping, rotation, cropping, shearing, and changing the contrast.

- **Early Stopping**

  It stops training when the validation loss gets higher than the training loss or the target metric (i.e. accuracy, AUC, etc.) does not improve. It assumes that if the validation loss does not improve, there is nothing to run. One can also stop training by setting the limited epoch predicting that overfitting will happen after the certain epoch.

- **Dropout**

  The term "dropout" refers to dropping out units (hidden and visible) in a neural network. These units are selected randomly and become invisible during the training phase. As illustrated in Figure 2.6 some neurons are cancelled out when dropout is applied. With the use of dropout, the model becomes more robust, resulting in a more generalized one. [10] Dropout probability decides how many neurons to deactivate. This parameter value is set between 0 and 1. Dropout of 1 means making all the elements zero, whereas dropout of 0 means applying no dropout at all.

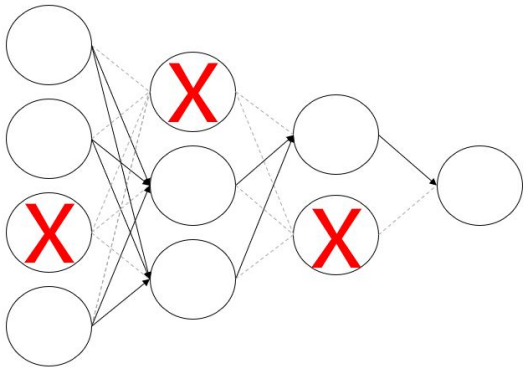

FIGURE 2.6: Dropout network. When applying dropout, neurons are randomly deactivated (dropped) by dropout probability. Neurons that are removed from the network are marked with red X. Dropout improves the network's performance by reducing overfitting.

2.1.7 Confusion Matrix

In order to verify the effectiveness of the trained models, we need evaluation metrics. For the classification problem, these metrics can be calculated from the confusion matrix.

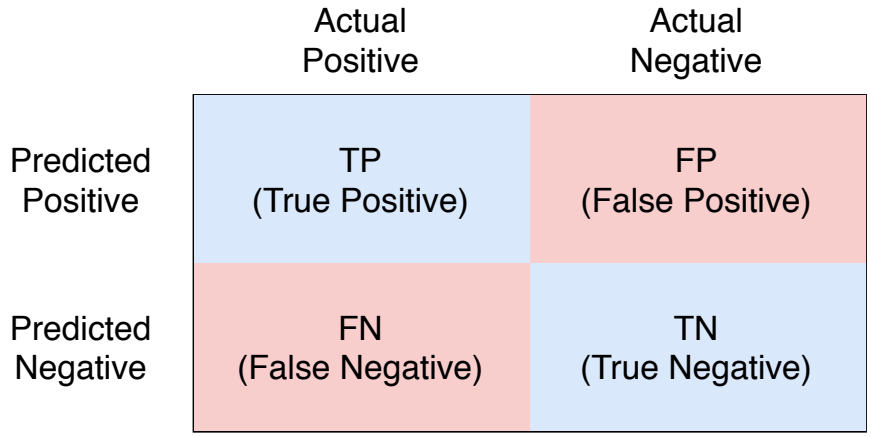

FIGURE 2.7: Confusion matrix of a binary classification problem. It is a 2 x 2 matrix that shows 4 possible results as well as t he 4 combinations with its corresponding class (positive or negative).

- True Positive (TP): When the actual label belongs to the positive class and the predicted class is also positive.
- False Positive (FP): When the actual label belongs to the negative class but the predicted class is positive.
- False Negative (FN): When the actual label belongs to the positive class but the predicted class is positive.
- True Negative (TN): When the actual label belongs to the negative class and the predicted class is also negative.

2.1.8 Evaluation Metrics

- **Accuracy**

  Accuracy measures the number of correctly classified outputs out of all the outputs.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.15}$$

  It is an effective way to evaluate the algorithm when the class of data is well balanced. If the data is class-imbalanced, accuracy alone cannot tell if the model performs well or not. Let us imagine a "Yes" or "No" classification task and the dataset of 9 samples from the label "Yes" and 1 sample from the "No" label. There is a model that classifies all the instances as "Yes" and we want to evaluate this model. The true positive will be 9 and the true negative will be 0, thus accuracy is 90%. However, this model is not a good model because it can never classify "No" labeled samples correctly. This is why it is recommended to use other measures such as precision and recall along with accuracy.

- **Recall**

  Recall measures a ratio of the correctly classified positive class to all the actually positive data samples.

$$Recall = \frac{TP}{TP + FN} \tag{2.16}$$

  This metric is suitable for health-related tasks such as cancer detection. Since it is better to classify a healthy person as a cancer patient than to classify a cancer patient as a healthy person and thus miss the diagnosis.

- **Precision**

  Precision measures a ratio of the correctly classified positive class to all the outputs that are classified as positive.

$$Precision = \frac{TP}{TP + FP} \tag{2.17}$$

  This metric is used when a task favors false negative over false positive which is the opposite case of the recall metric. In other words, it is useful for a task when the occurrence of false positives is not tolerable. For instance, the algorithm needs to classify whether the product is in high quality or not for a shop owner. If a bad quality product is classified as good (false positive), the customer will complain and the reputation will go down (high cost).

- **F1-score**

  F1-score or F-measure is the harmonic mean of precision and recall. [11]

$$F1 - score = \frac{2 \times recall \times precision}{recall + precision} \tag{2.18}$$

  F1-score measures how balanced the precision and recall are. Thus, it can be used as a complementary metric to accuracy especially when the data is not

evenly distributed. When an F1-score is high, (the possible maximum value is 1 and the possible minimum value is 0), it indicates good precision and recall.

- **Specificity**

  Specificity measures a ratio of the correctly classified negative class to all the outputs that are actually negative.

$$Specificity = \frac{TN}{TN + FP} \tag{2.19}$$

  It is recommended to use this metric when the occurrence of false positives has a high cost and true negatives are needed. For instance, a polygraph test should not have false positives. Instead, it should detect true negatives well. In disease detection problems, specificity is useful for ruling in unhealthy patients. When the specificity is high and the patient has positive results then this patient is most likely to have a disease.

- **ROC-AUC Score**

  The ROC (Receiver Operator Characteristic) -AUC (Area Under ROC Curve) Score measures how well the model can distinguish between classes. [12] In cancer detection problem, high AUC means that the model can distinguish better between patients with cancer and healthy patients.

2.2 Convolutional Neural Networks

Convolutional Neural Networks(CNN) is one of the most widely used deep learning networks. It has shown excellent performance in the pattern recognition fields such as image processing, and voice recognition.

A CNN architecture consists of an input layer, output layer, and several hidden layers. In the basic CNN's hidden layers consist of convolutional layers, pooling layers, fully connected layers, batch normalization layer and activation layers (such as ReLU and ELU). A convolution layer is a layer that contains filters that convolve the input and extract features. A pooling layer is a layer that performs pooling and it usually is located between two consecutive convolutional layers. After going through pooling layers, the dimension of the network is reduced. A fully connected layer performs the final classification and is positioned before the output of the network. Batch normalization layer standardize outputs of the previous layer by applying normalization. Figure 2.7 presents a CNN architecture that is used in this work.

Convolutional layers detect patterns such as edges, shapes, and corners in images. This detection process is done by filters. A filter can be seen as a $n \times n$ matrix. To decide how the filter convolves the input image, we need stride and padding: stride is the number of steps that the filter is going to take, while padding is the extra pixels that are added in the surrounding of an input.

The filter moves around the input. First, it moves from the left to the right then when it reaches to the right-most column it goes down till it reaches the bottom-right. The convolution calculation is done by element-wise dot product. Both stride and padding determine the size of the output. Figure 2.8 represents an example of a basic convolution calculation. The filter is 3×3 with stride 1 and padding 0.

Different padding sizes determine the output size as you can see in figure 2.9. All three cases use a stride of 1. When the padding is valid, it does not add any padding

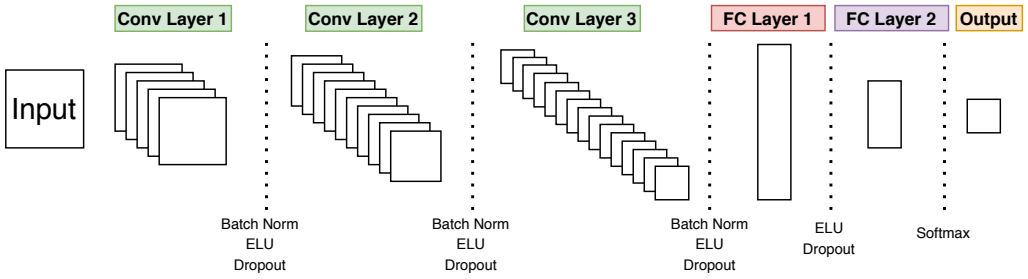

FIGURE 2.8: A CNN architecture example. Input goes through three Conv Layers (convolutional layer) and afterwards to 2 FC Layers (fully connected layer). Between each Conv Layer and after the last Conv Layer (Conv Layer 3), batch normalization (batch norm), ELU activation function and dropout are applied.

to the input image. As a result, the output size becomes smaller than the input size. When the padding is the same, the padding of 1 is added to the input image. The output size is as same as the input size. The last case occurs when the padding is full. The padding of 2 is added to the input and the output size is bigger than the input size.

Max Pooling is one of the pooling forms. As the filter convolves the input, it calculates the maximum value. As a result, the output has a smaller dimension than the original input. Figure 2.11 illustrates a max pooling with 2 x 2 filter with stride of 2. The filter convolves the input in the following order (left to right):

$$\begin{bmatrix} 3 & 5 \\ 0 & 7 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 9 & 3 \end{bmatrix} \begin{bmatrix} 11 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 2 & 7 \\ 9 & 4 \end{bmatrix}$$

Then the maximum value of each section is selected, resulting the 2 x 2 output: $\begin{bmatrix} 7 & 9 \\ 11 & 9 \end{bmatrix}$ As max pooling reduces the number of parameters, it can ease the over-fitting problem and have less computational cost. [13]

2.3 Medical Data

"Medical imaging is one of the most valuable sources of diagnostic information but is dependent on human interpretation and subject to increasing resource challenges." [14] According to the study conducted by Liu et al [15], it is shown that the deep learning algorithms can reach the equivalent performance to health-care professionals (radiologists) although there is still a limitation of integrating these algorithms to the real-world practices. Liu et al claim that deep learning algorithms for medical imaging perform as good as, or even better than, the health-care professions. This gives validity to invest and study more on improving algorithms for medical imaging. Medical imaging deep learning can be used to make AI-based medical equipment and this could be a cooperative tool for health-care professions.

2.3.1 Cancer

According to WHO [16], "cancer is a generic term for a large group of diseases that can affect any part of the body. Other terms used are malignant tumors and neoplasms." Tumors develop when cells that reproduce too rapidly are accumulated.

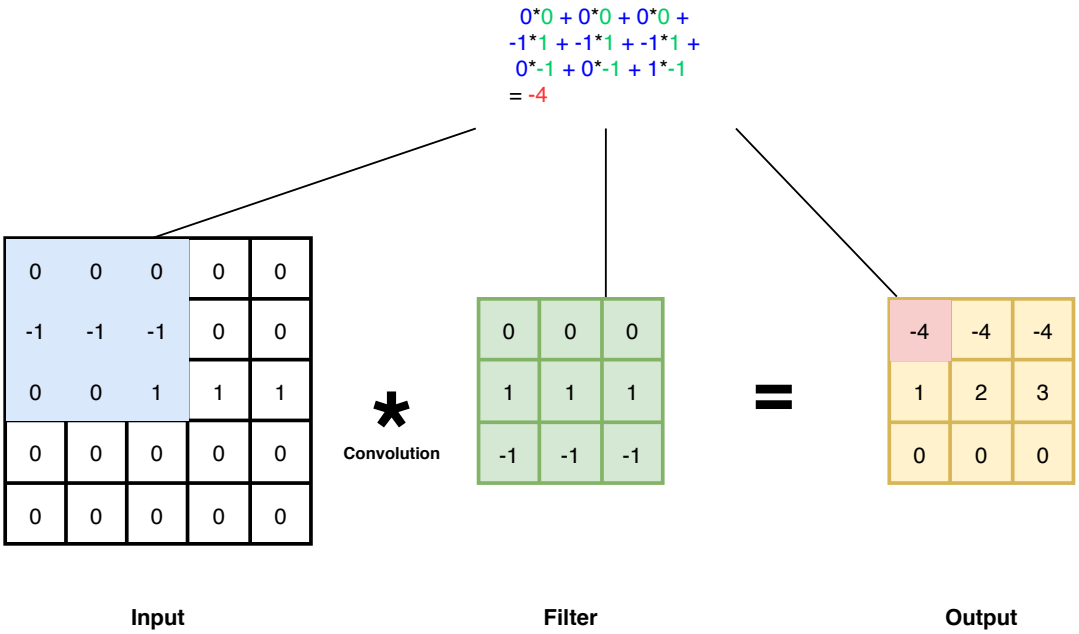

FIGURE 2.9: A basic convolution calculation. The 5 x 5 input matrix is calculated with a 3 x 3 convolution filter. The example of the convolution calculation is presented above. The result of the convolution calculation is 3 x 3 output matrix which is shown on the right.

With the help of medical imaging, cancerous tumors can be detected autonomously. The main types of tumor are:

- Benign: When the cells are not cancerous.
- Premalignant: When abnormal cells that could be cancerous are found.
- Malignant: When the cells are cancerous.

2.3.2 Data Types

There are many medical imaging types. In cancer detection, MRI, CT scans, and mammograms are most widely used.

- **MRI**

  Magnetic Resonance Imaging generates three-dimensional images of body parts using magnetic fields. Depending on MRI sequences, images can have different appearances. T2 weighted, PD (Proton Density weighted), DWI (Diffusion weighted images), ADC (Apparent diffusion coefficient), DCE (Dynamic contrast enhanced) and K-trans are examples of MRI sequences. [17] The sequences that are used in this work are T2 weighted, DWI, and ADC grayscale images. When describing the shade of MRI images, the word "signal intensity" is used. High, intermediate, and low signal intensity refer to white, gray, and black shade respectively. In T2 weighted and DWI images, tumors have high signal intensity whereas they have low signal intensity in ADC images.

- **CT scans**

  A computed tomography scan produces three-dimensional medical images using X-rays from different angles. It is a combination of X-ray images that are

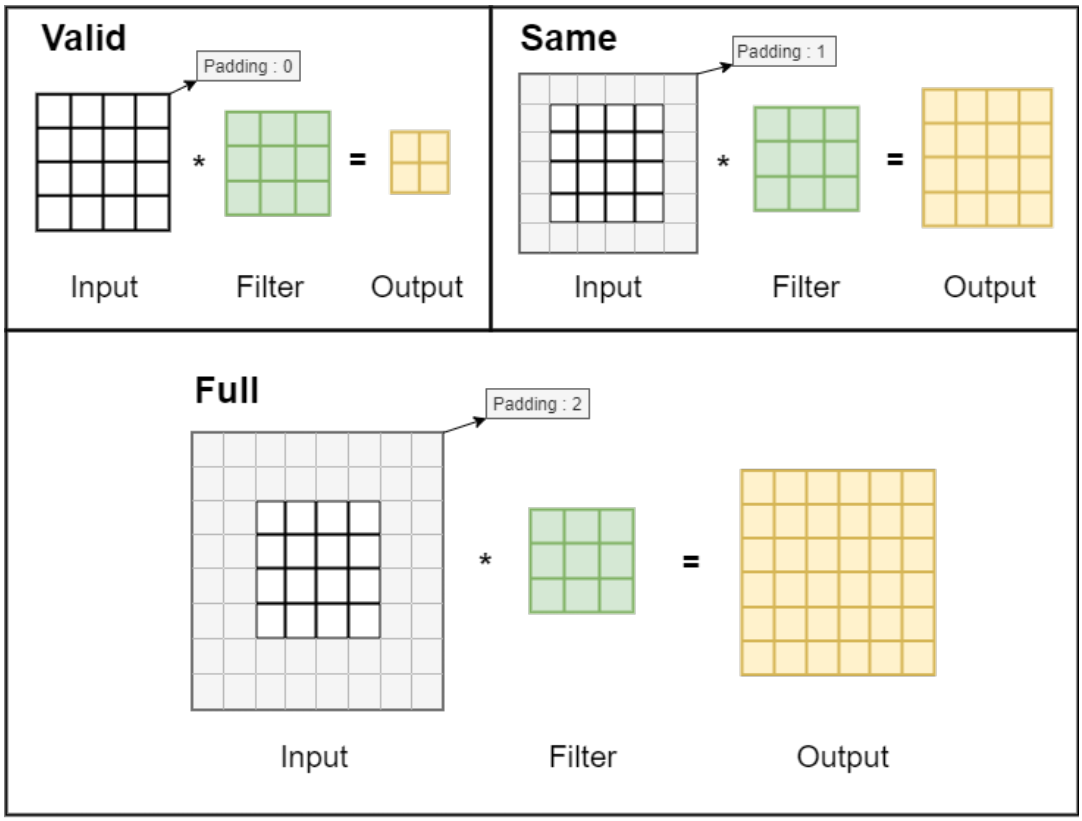

FIGURE 2.10: Different types of padding and consequent output size. Each convolution process is shown. In the case of "valid" no padding is applied, making the output smaller than the input. For "same", padding of 1 is applied and the input is surrounded by one extra pixel. The output size is the same as the input size. For "full" type, padding of 2 is added to the input. After convolution, output size becomes larger than the input size.

taken from different angles of a body. Figure 2.15 shows an example of a lung CT scan image.

2.3.3 DICOM

The DSP and DSL are in DICOM file format with .dcm extension. DICOM (Digital Imaging and Communications in Medicine) is a standard medical imaging format. It is widely used to store and exchange medical images such as MRI, CT scans. DICOM files contain tags from which you can access to important information about the patient. With the patient's information, we can connect the image to each patient.

2.3.4 Manipulating DICOM images

DICOM files have to be in a consistent representation before being converted to other formats for the physicians to detect anomalies accurately. To this end, pixel data is normalized, using the following tags:

- Instance Number: The order of the images.
- Pixel Data: The values of the pixels.

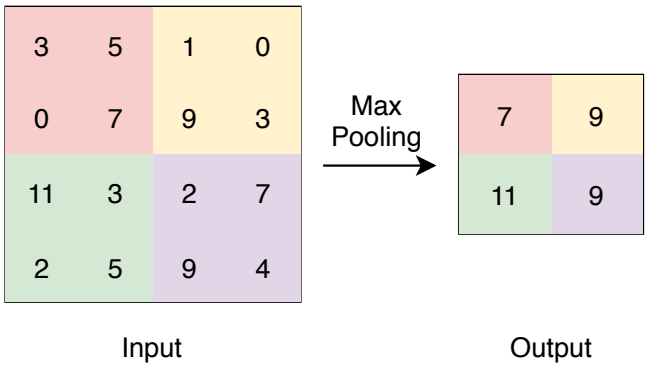

FIGURE 2.11: Max pooling with 2 x 2 filter, stride of 2. The left matrix represents the input before the max pooling. The right matrix shows the result of the max pooling.

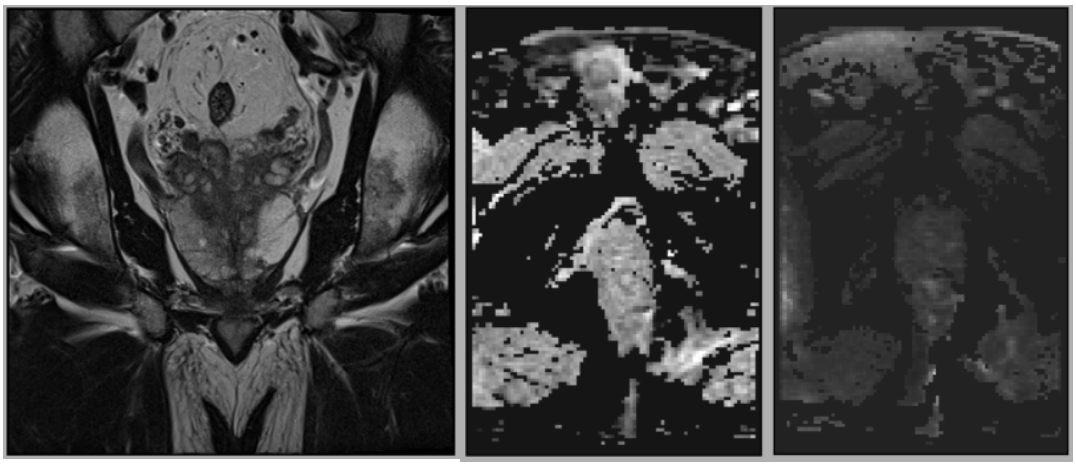

FIGURE 2.12: Different sequences of a prostate MRI image. Tumors have white shade in T2-Weighted images and ADC images. In DWI images, tumors have black shade.

- Samples per Pixel: the number of channels for each pixel
- Bits Stored: The number of bits stored for each pixel sample

Hounsfield correction and a linear transformation are applied to convert pixel data to the correct numpy arrays. The former is a measure of radio-density, calibrated to distilled water and free air. [18] First, the pixel values go through the Hounsfield Units (HU) correction:

$$HU = m \cdot P + b \tag{2.20}$$

The equation involves using the rescale slope m and rescale intercept b. P refers to the pixel value. After HU correction, a linear transformation is then applied.

$$\text{if } (P \leq c - 0.5 - \frac{w-1}{2}), \text{ then } y = y_{min} \tag{2.21}$$

$$\text{else if } (P > c - 0.5 + \frac{w-1}{2}), \text{ then } y = y_{max} \tag{2.22}$$

$$\text{else } y = (\frac{P-(c-0.5)}{w-1}) + 0.5) * (y_{max} - y_{min}) + y_{min} \tag{2.23}$$

"where c is the window center, w window width, P the pixel input value, y the pixel output value, y_min the minimal value of the output range, y_max the maximal value of the output range." [6] These equations make sure that the pixel values are distributed correctly within the range between 0 (the minimum output value) to 255 (the maximum output value)

2.4 Dataset Description

The DSP (ProstateX Challenge Dataset) consists of multi-sequence MRI images. The DSB (Kaggle Brain Dataset) is in single-sequence MRI format. The DSL (Lung CT Challenge dataset) contains CT scans. Since the DSB and the DSL do not have multi-sequence exams, extracting single channel images from the DSP was necessary. The DSP and DSL are converted from DICOM to Numpy arrays. The DSB is converted from JPG to Numpy arrays. Each of the datasets are resized to the same dimension and normalized for a consistency purpose. Before re-scaling the images, cropping or manual removal of unclear images are applied respectively. Different augmentation factors are applied to each dataset, depending on how imbalanced the dataset is. All the detailed steps are explained below.

- **DSP: ProstateX Challenge Dataset**

  The DSP consists of 25314 training patients, 2276 validation patients, and 329 test patients. The original prostate dataset is available in the official prostateX challenge website [19]. The original prostateX challenge dataset were augmented and processed. To make it consistent with other datasets, single channel images have to be extracted from the multi-sequence images. DWI images are chosen due to their resemblance to CT scans. All the DICOM files are converted to NumPy arrays and the size of DWI images are scaled to fit that of the T2-weighted images, since the DWI image sizes are not consistent. After aligning the images, they are resized to 65x65px using a bicubic interpolation after cropping the images to 130x130px. With these dimensions, most of the lesions with a small margin of a border around them can be covered, which made the detection process easier. Then, z-score normalization (see equation 2.1) is applied across the entire images.

  The original dataset has a class imbalance problem as the ratio of the false and true class was 3 to 1. To solve this class imbalance problem, two approaches are proposed. The first one is downsampling the size of the data which belongs to the class of a bigger size. The other one is augmenting each class using different augmentation factor. The authors choose the latter, since the amount of the available dataset is quite small. The true class dataset is augmented by the factor of 60 which is bigger than that of the false class dataset's to make the dataset proportional.

  Figure 2.13 illustrates MRI images with different sequences of the DSP. Tumors usually appear white in T2-weighted images and DWI images. In ADC images, they are shown in black. DWI images are selected, since it has the same tumor color as CT scans. T2-weighted images are only used to ensure the same alignment and size of the other images.

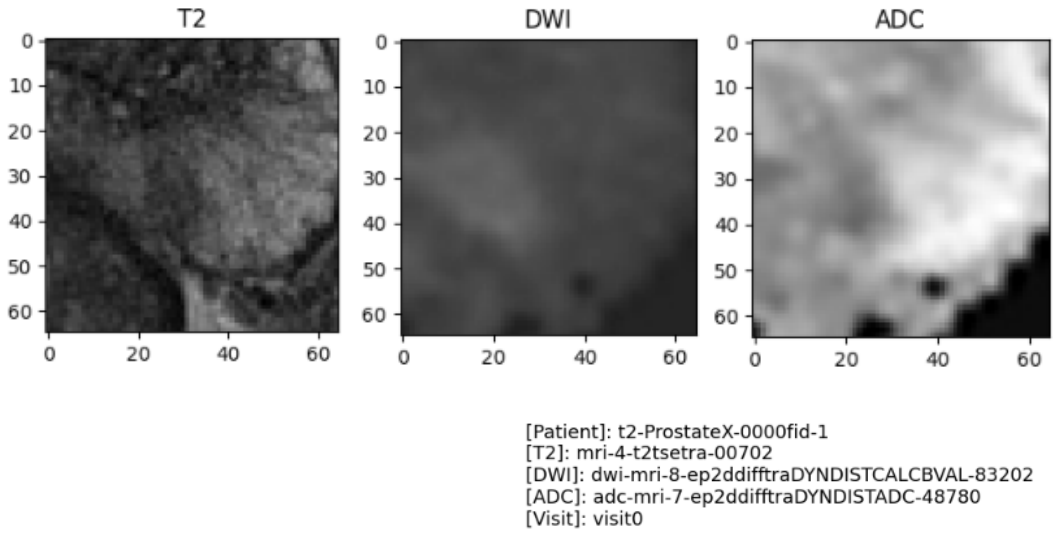

FIGURE 2.13: Example of the DSP. An MRI image with different sequences, T2-weighted (T2), DWI, and ADC, are presented. In T2 and DWI images, cancerous nodules are shown as white, whereas they appear as black in an ADC image.

- **DSB: Kaggle Brain Dataset**

  The brain dataset consists of 36000 images for training, 4800 images for validation, and 242 images for test. The brain MRI images are publicly accessible from Kaggle [20]. The dataset consists of single-sequence MRI images. Low quality images and images with disturbing objects are manually removed. Since it is only given with the clinical significance, a ground truth CSV file have to be created from scratch. First, they unified the file names and formats which is "[yes|no]index.jpg". A few PNG files in the dataset are converted to JPG files. In the CSV file, columns "PatientID", "Nodule Center Position", "fid", and "Diagnosis were added.

  - PatientID: The filename without extension
  - Nodule Center Position: The pixel coordinates of the lesion
  - fid: Finding ID
  - Diagnosis: Clinical significance (True or False)

  The tumor sizes in the dataset are inconsistent. The patch size which could fit the large tumors is selected. Then, the images are resized to 65x65px using a bicubic interpolation. After converting the images to JPG format and resizing them, the images are exported as NumPy array for normalization purposes. The z-score normalization is applied (equation 2.1). The classes of the brain dataset are more or less balanced, so they are augmented by the same factor (200x). They do not include test patients of brain dataset since their target dataset is the prostate one.

  According to Clement et al's experiment [6], cropping the lesions with more borders and margins are suitable for single-sequence images. On the other hand, for multi-sequence images, cropping the images to fit more or less perfectly help the detection process better.

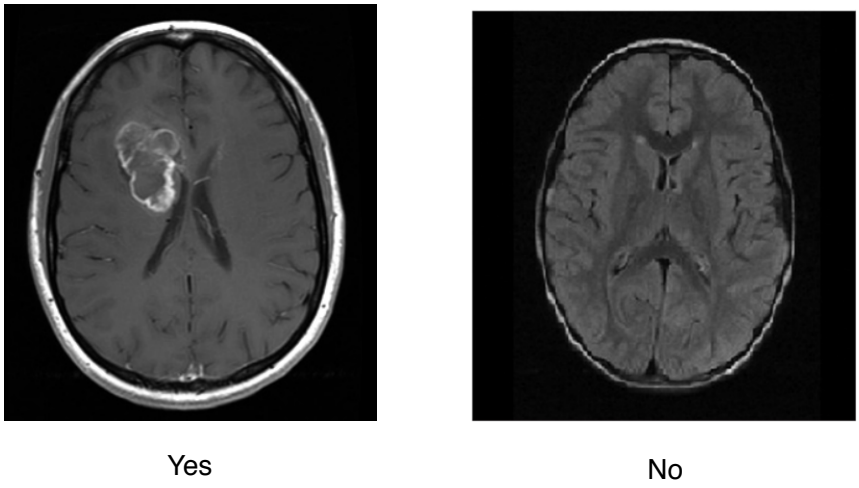

FIGURE 2.14: Example of DSB. The left picture shows a brain MRI image with a malignant tumor in it. The tumor is shown as white shade on the left side of the image. The right picture is a brain MRI image that does not contain a cancerous tumor.

Figure 2.14 shows an example of the DSB. The left one is a healthy patient's brain image and the right one is a brain image with a cancerous tumor. In this example, tumor shows up as white on the middle left side of the image.

- **DSL: Lung CT Challenge Dataset**

  The DSL is publicly available from SPIE-AAPM Lung CT Challenge website [21]. Two CT scan sub-datasets (calibration set and test set) are merged due to the lack of data. Like the brain dataset, two CSV files (Testset.csv and CalibrationSet.csv) are made from the two originally provided CSV files to better handle the information. Since the labels of the dataset are not binary, they have to be divided into positive and negative.

  - Positive: "malignant", "Primary lung cancer"
  - Negative: "benign", "Benign nodule"

  Two images with the label called "Suspicious malignant nodule" are removed since the diagnosis is unclear. All the images in the lung dataset have the same resolution of 512x512px, which makes aligning the images as DSP unnecessary. However, the size of tumors varies greatly and the boundary of tumors is ambiguous. The patch size is therefore set according to the size of the large tumor. Images with excessively large tumor sizes are considered as outliers and removed. The images are cropped to 85x85px and then, finally resized to 65x65px using a bicubic interpolation. Finally, DICOM files are converted to NumPy arrays. Like the brain dataset, the same augmentation factor (240x) is applied since the lung dataset is roughly balanced. It consists of 14120 images for training, 2640 images for validation, and 77 images for test.

  Figure 2.15 illustrates a CT scan image of a cancerous tumor in it. Black, gray and white parts are observed in the image. Tumor are usually shown as white in CT scan images.

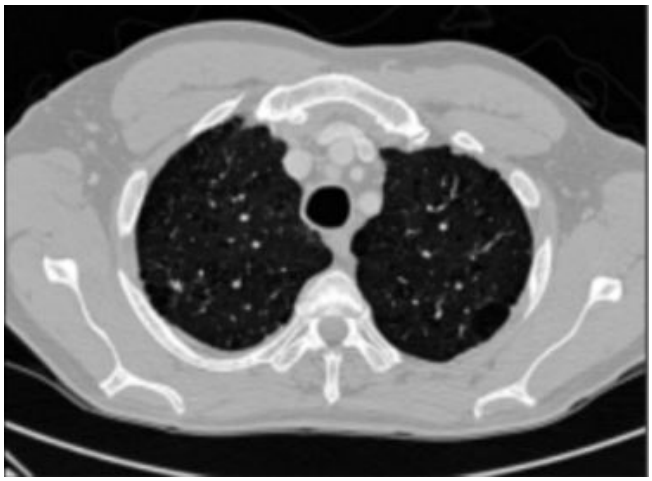

FIGURE 2.15: Example of DSL. The image shows a lung CT scan image that has a cancerous tumor. Usually, a normal brain appears as black. Nodules are shown as white spots.

2.5 Related Work

- Prostate

  According to the World Cancer Research Fund International [22], Prostate cancer is the second most common cancer worldwide, and the fifth most common cause of cancer death among men in 2018. Early detection can increase the survival rate of prostate cancer patients.

  Yoo et al [5] proposed a pipeline that uses ResNet as their base architecture. The proposed pipeline consists in 3 stages. First, 5 CNNs models give probabilities of each class (cancer and non-cancer) for each DWI slice. Then, the feature selector selects key features from the first-order statistical features which were extracted from the outputs of the first stage. Finally, a Random Forest classifier classifies patients into two classes using the features from the previous stage. The dataset consists of 427 patients in DWI images and was divided into a training set (64%), a validation set (11%), and test set (25%). The DWI images with 6 channels (ADC, b0, b100, b400, b1000, and b1600) were used as input, and each image was resized by 144 x 144 pixels and cropped by 66 x 66 pixels so that the images include the prostate region. Z-score normalization was applied to all of the data which is

$$P_{i_normalized} = \frac{P_i - mean}{std} \tag{2.24}$$

  P_i represents the pixels of a slice and std means standard deviation of the slices. The proposed CNNs model consists of convolutional layers with 7 x 7 filter, 3 x 3 max pooling layer, 2 ResNet Blocks, an average pooling layer, and a fully connected layer. In the first ResNet Block, 3-layer bottleneck blocks which are composed of CNN layers with filter sizes 64, 64, and 256 are stacked 4 times. In the second ResNet Block, 3-layer bottleneck blocks which consist of CNN layers with filter sizes 128, 128, and 512 are stacked 9 times. They chose Stochastic Gradient Descent (SGD) as the optimizer and binary cross entropy as the loss function because of the class-imbalanced dataset. In slice-level performance, their best CNNs among 5 individually trained CNNs models achieved AUC

of 0.87 in 95% confidence interval. In patient-level performance from the last stage's result with the Random Forest classifier, their best AUC was 0.84 in 95% confidence interval.

- Brain

  According to Farmanfarma [23], brain cancer is one of the most common cancers in the world and the incidence of brain cancer is increasing. It is reported that the mortality rate is about 3.4 per 100,000 people in the world.

  Saxena et al [24] compared three pre-trained CNN models (VGG-16, Inception-V3 and ResNet-50) using transfer learning. They used the Kaggle brain dataset and the dataset was split into 183 training sets, 50 validation sets, and 20 test sets. The entire dataset was normalized and augmented. To make the data suitable for the pre-trained model, the images were resized as 224 x 224 x 3 dimensions. Among all the three CNN models, VGG-16 and ResNet-50 could achieve high AUC of 0.9 and 0.95 respectively.

- Lung

  According to American Lung Association [25], five-year survival rate of lung cancer is only 18.6 %, however, when detected in an early stage with the help of low-dose CT screening, the mortality can be reduced by 14 % to 20 %. This emphasizes that detecting lung cancer at an early-stage is a key to the survival of a patient.

  Cengil et al [26] used the SPIE-AAPM Lung CT Challenge dataset and proposed a three-dimensional CNN network. The network consists of 7 layers (Input layer, five three-dimensional CNN layers and fully connected layer). The input layer takes a four-dimensional tensor where an additional dimension "depth" is added to the height, width, and channel parameters. They chose accuracy for their evaluation metric and achieved 0.7 of accuracy.

Chapter 3

Transfer Learning

This chapter is devoted to transfer learning and a related work "Prostate Cancer Classification: A Transfer Learning Approach to Integrate Information From Diverse Body Parts" by Clement et al [6]. A brief explanation of the training process of the work is presented. A concept of transfer learning is introduced and, later in Chapter 4, it is compared with multitask learning.

3.1 Transfer Learning

Transfer Learning is a learning technique where the model is developed for one task and is applied to a similar, but different tasks.

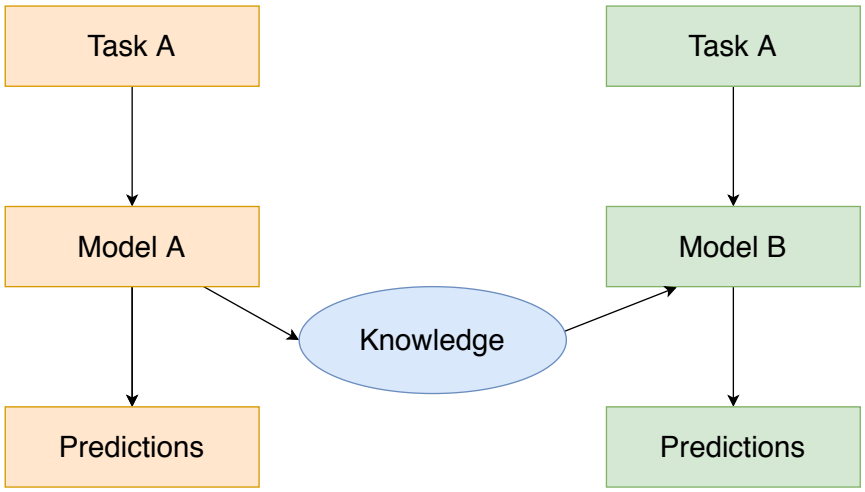

FIGURE 3.1: Transfer learning process. The left column shows a traditional machine learning where task A is trained with the model A. The right column represents a transfer learning where the knowledge gained from training task A is passed on to the model B.

Figure 3.1 illustrates the transfer learning process. Knowledge gained by training a model A (usually on a large dataset) can be transferred to task B so that the task B can take advantage of the pre-trained model A's ability to detect common features. For example, the model trained to classify animals can be applied to another model whose goal is to classify a specific animal.

This learning method is useful when:

- The different datasets share similar features.
- There is not enough data to train.

There are some major advantages in using transfer learning. It gives an important benefit when it comes to the initial training. It can give a good start because the model has already learned common low-level features. Another advantage is that, during the training phase, the rate of improvement of the model becomes steeper. Lastly, the trained model converges better, which means the loss can reach the minima better. [27]

3.2 Transfer Learning Application Review

This section gives an overview of the new transfer learning approach by Clement et al. [6]. The work improves prostate cancer detection by introducing a new learning algorithm. The work proposes a new algorithm based on transfer learning to improve accuracy. The main limitation that medical imaging application has, is the amount of available data. It is hard to acquire suitable medical data due to ethical problems. It also has to go through an anonymization process and needs to meet strict security conditions. Transfer learning is an alternative method to tackle the problems mentioned above. The learned knowledge is transferred to other similar tasks and the tasks can have a benefit from the pre-trained model. The goal of the work is to classify prostate cancer. The work gathered similar medical images of different body parts, which were brain and lung images. By using the brain and lung's features, their model could generalize and perform better on classifying prostate cancer.

3.3 Process Overview

First, the dataset for each body part is gathered, and preprocessing is applied. The work trains the model on the target dataset (Prostate dataset) and save the model which has the highest AUC. Then, the work takes two different steps on non-target datasets (Brain and Lung dataset);

1. Only the feature extractor from the previous step is loaded and the decision maker is reset. The feature extractor is frozen and only the decision maker is trained. The term "freezing" refers to keeping the weights when loaded and not updating them. "Unfreezing", on the other hand, has to do with making the layers update the weights when trained. After all the epochs, the best performing model is saved.

2. The model which had the best AUC and accuracy is loaded and the feature extractor is unfrozen. End-To-End training is applied to the same dataset. End-To-End training means to train the undivided model (the feature extractor and the decision maker).

After the above two steps are performed on non-target datasets, the best model is saved and loaded. The model is trained again with the frozen step on the target dataset. Here, instead of resetting the decision maker with random initialization, the decision maker from the first step, which has the highest AUC, is loaded and attached to the model. Finally, the feature extractor is unfrozen and the whole model (the feature extractor and the decision maker) is trained on the target dataset. The best performing model with the highest accuracy and AUC are saved and tested on the target test set. Each step of training the datasets with freezing and unfreezing is done with 2000 epochs. [6]

3.3.1 Evaluation metric

The evaluation metric used for the transfer learning implementation is AUC (Area Under Curve). AUC is a frequently used evaluation metric, especially in medical imaging deep learning applications. AUC measures the prediction ability of a model. For a cancer detection model, the model with a higher certainty of its prediction is preferred.

3.3.2 Dataset 1 (Prostate)

From 200 random initialization, the best initialization which has the best AUC on the prostate validation dataset is chosen. The whole dataset is trained with a learning rate of 1e-8, and a dropout rate of 0.4. The gap between the training and the validation graphs is not so large, which means that model is learning features well and also generalize on the unseen data.

3.3.3 Dataset 2 (Brain)

The best model from the previous step is saved and loaded when training the brain dataset. Then, the feature extractor is frozen and the decision maker is initialized with random weights. At the beginning of training, the accuracy and AUC are fairly low because of the random initialization in decision maker layers. The model conducts a classification task solely based on a feature extractor from the previous step. This shows that the features extracted from the prostate dataset adapted well to the brain cancer dataset. Furthermore, training and validation curves behave similarly with a little gap in between. After the frozen process, the feature extractor layers were unfrozen and the end-to-end model was trained on the same brain dataset with a different learning rate of 1e-8. Since the decision maker is trained beforehand with a feature extractor which was suitable for the brain dataset, the initial metric values has a high starting point. During the training, the model focuses on learning new features from the brain cancer set.

3.3.4 Dataset 3 (Lung)

The previous model is attached and the feature extractor is frozen. Like the previous training, the decision maker is randomly initialized again. Accuracy and AUC from the frozen training was lower than the brain dataset training, however, the training and validation graph still shows a good result. It indicates that the features from the prostate and brain are helping the model to learn better. For the full end-to-end training step, the model is trained with a decreased learning rate.

3.3.5 Dataset 4 (Prostate)

For the final step, the target dataset (prostate) is trained again with the frozen feature extractor from the previous step and the decision maker from the first step of full training on the prostate dataset. Now, the last layer is trained again with the help of features learned from the other datasets (brain and lung). The high values of the metrics in the beginning show that the transfer learning worked well among all the three datasets. The model slowly improves thanks to the common low-level features that were acquired from the brain and lung dataset. Finally, the frozen layer is unfrozen, and full training is performed. The model reached its optimal at around 500 epoch. For this step, dropout was omitted, however, the values of the metrics of

validation set were higher than those of the training set which suggests the model could generalize well. Since the training was for prostate cancer detection, the best model was tested on a test set. The best model could get an accuracy of 0.6842 and AUC of 0.8000 which were more or less equivalent to a novice radiologist's result.

Chapter 4

Multitask Learning

This chapter presents an explanation of what multitask learning is. A description of what it is and differences with the traditional machine learning approach are given. There also exists some differences between multitask learning and transfer learning. The comparison between these two learning algorithms are also looked at.

4.1 Multitask Learning and Single task Learning

Traditional machine learning focuses on building and optimizing a model for one specific task. While we still can get a reasonable result, helpful information from training signals of related tasks might be missed out. Multitask learning is an alternative machine learning approach to exploit such information. Multiple related tasks are trained at the same time using shared hidden layers. It aims to improve generalization performance for one main task. [28] This algorithm is based on the idea that tasks can share what they learn via a shared common hidden layer and are trained in parallel. [29]

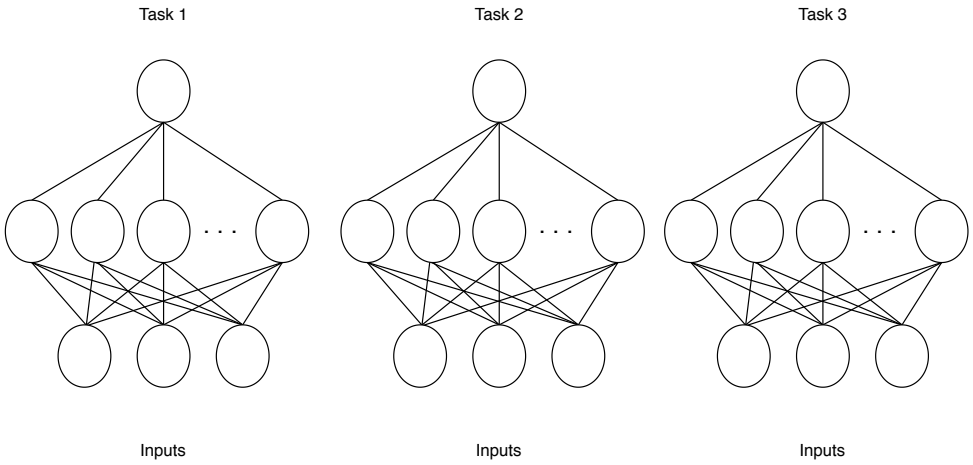

FIGURE 4.1: Traditional single task learning. Three machine learning models are presented. Each task is considered as an independent model, thus trained separately. They do not share any layers. [28]

Figure 4.1 shows several single task learning (STL) neural networks. Each of the neural network is given the same inputs, but different tasks. The networks are trained individually, not sharing any hidden layers with each other. Figure 4.2 shows a multitask learning (MTL) neural network. Unlike single task learning, MTL has a hidden shared layer that is shared by all the tasks. The tasks consist of the main task and extra (auxiliary) tasks.

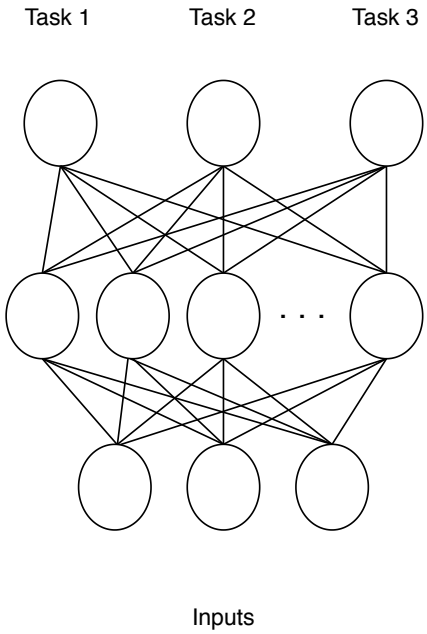

FIGURE 4.2: Example of multitask learning. The task 1, 2, and 3 are different tasks but they are trained at the same time. All the tasks are trained on the same input. Three tasks share hidden layers. [28]

4.2 How does MTL work?

Traditional machine learning techniques focus on learning a single task at a time. However, MTL is a learning algorithm that trains multiple tasks. When the multiple tasks are learned simultaneously, the model can learn better with improved generalization. The reason why the model learns better in MTL is due to the inductive bias helping the shared hidden layer.

There are some possible explanations of why Multitask Learning works. First, when the tasks are not correlated in the backpropagation stage, the network might take added gradient as noise to other tasks. [30] This can improve generalization. The second possible reason is that added tasks might change parameter updating dynamics which supports networks with multiple tasks. The last possibility is that, thanks to the reduced net capacity, the model is better generalized. Caruana et al's work [28], shows that this is achieved through the additional training signals work as domain-specific inductive biases, not because of the above reasons. They shuffle the extra task training signals among all the cases in the training set, which gets rid of the relatedness of the main task and the extra tasks.

4.3 Related tasks

A task in machine learning is a formulation of what we want to predict based on the available data. A task can be divided into the main one and the extra one. Typically, the extra tasks are a subset of the main task. For instance, if the main task consists of recognizing whether the object is a cat or a dog, the extra tasks would consist of recognizing the shape of the ear and the color. To utilize multitask learning, a main task and extra tasks need to be related. As caruana stated [28], "MTL is an inductive transfer method that uses the domain specific information contained in the training signals of related tasks."

An inductive transfer is a learning technique which makes use of information gained from other similar tasks to improve generalization. MTL is considered as an inductive transfer, because it leverages supplemental information from the training signals of related tasks. This information from similar tasks works as an inductive bias to improve the generalization of the main task. From the perspective of the main task, the additional tasks are viewed as bias.

4.4 MTL Mechanism

Caruana [28] suggested various mechanisms to explain how the backpropagation nets can generalize better. For all the cases below, the notation T refers to a task, T' refers to a task related to T and F refers to a hidden layer feature.

- **Statistical Data Amplification**

  MTL has an effect of increasing a sample size thanks to the information in the training signals of extra tasks. This is called data amplification. When the noise is added to T and T' respectively, a net gains additional information which can be seen as amplification of data. If the two tasks T and T' can notice that they share F, then a model learning the two tasks averages F through different noise processes. As a result, the hidden layer F can be learned better.

- **Attribute Selection**

  A model with few data or significant noise which is fitted to only learn the task T is not likely to well distinguish inputs pertinent to F from ones that are not relevant to it. However, a model that learns both T and T' will better determine more relevant inputs to use to train F thanks to improved training signals for it.

- **Eavesdropping**

  Let us consider that there is a hidden layer F and two tasks T and T' that are not learned together. A net can learn F well when learning the task T, but a net learning T' cannot learn F easily. However, when T and T' can be learned together in an MTL net and they can share the hidden layer F, then the net learning T' can obtain information by observing the F learned for T (eavesdrop). Due to the additional information that T' learned about F by eavesdropping, a model can learn F better.

- **Representation Bias**

  When there are two tasks T and T', the MTL model is biased to select hidden layer representations that both tasks prefer. The MTL model also tends not to select the representations that other tasks do not prefer. [28]

4.5 Parallel and Sequential Transfer

4.5.1 Parallel Transfer

When the information learned for one task is transferred to another task when being trained, it is called parallel transfer. Each tasks are learned at the same time, while transferring their knowledge to each other.

MTL uses parallel transfer. According to Caruana [28], the advantages of parallel transfer are:

- The full detail of what is being learned for all tasks is available to all tasks since all tasks are being learned at the same time.
- In many applications, the extra tasks are available in time to be learned in parallel with the main task(s). Parallel transfer does not require one to define a training sequence. The order in which tasks are trained often makes a difference in a serial transfer.
- Tasks often benefit each other mutually, something a linear sequence cannot capture. For example, if task T is learned before task T', task T' cannot help task T. This not only reduces performance on task T, but can also reduce task T's ability to help task T'. [28]

4.5.2 Sequential Transfer

Sequential transfer learning indicates that a model is learned for one task and the learned knowledge is transferred to another task sequentially (one after another). The whole training of a model happens first then the knowledge is transferred to another task.

In the case of transfer learning, the pre-trained model is used as a baseline for a new task. The tasks are learned sequentially. A model trained to learn a task T can be used as a base for a model made for a task T', since it is not learned at the same time. The two nets cannot help each other at the time of learning. Caruana et al [28] proposed an approach that MTL benefits from the pre-trained model. When the training data is not available from the previously learned model, synthetic data can be generated using the model and the training signals in the synthetic data can be used as extra MTL tasks.

Chapter 5

Proposed Approach

This chapter presents the whole process of the P-Hydra learning technique. The P-Hydra learning implementation partially relies on Clement et al's work [6] in terms of the data preprocessing and the model architecture. The difference is that instead of sequential transfer, we make use of parallel transfer, implementing a new P-Hydra algorithm.

First, the difference between multitask learning and our method is discussed. Then, the individual steps of the process are explained in the process overview alongside with the visual representations. Thereafter, the detailed information about the P-Hydra net that we use in this work is analyzed.

5.1 Different Aspects From Multitask Learning

The original multitask learning aims for the specialization of one main task, while our proposed method aims for generalization. Our goal is to have a generalized base for any medical imaging application. It can be used as a base (i.e. feature extractor) for cancer detection model so that only the training a decision maker part is needed. The main difference comes from the relationship between the main task and extra tasks. In multitask learning, the extra tasks are regarded as sub-tasks which are in the same domain, while our case, there is no distinction between the main task and the extra task. All the tasks are treated more or less equally. Caruana et al [28] defined the tasks as related when they are the same function of the inputs, but with independent noise processes added to the task signals.

5.2 Goal

Transfer Learning and Multitask Learning are particularly useful when the amount of data is not sufficient. In this work, we implement a new learning technique that is based on multitask learning to improve cancer detection application. The final goal of this work are:

- To prove that in cancer detection, our learning algorithm is valid
- To provide a baseline (the generalized feature extractor) for other cancer detection tasks.

To successfully transfer the knowledge and make it useful, the datasets must share common features. The training makes use of related three different datasets simultaneously. The premise is that DSP, DSB, and DSL that are used in this work share these common low level features. This was shown in Clement et al [6], as they improved the AUC of prostate cancer detection through sequential transfer.

Based on this result, we could apply the relatedness of these datasets to our P-Hydra implementation.

Dataset	Original	After Augmentation	Training Set	Validation Set	Test Set
DSP	1150	27919	25314	2276	329
DSB	226	41042	36000	4800	242
DSL	81	16837	14120	2640	77

TABLE 5.1: The number of training, validation, and test set for each dataset. Column "Original" represents the number of dataset before augmentation. Notice that the size of dataset has changed after augmentation.

5.3 Model Architecture

The model used here is based on a VGG-16 network. The model is divided into two parts: the feature extractor and the decision maker. The feature extractor collects the features that represent a cancerous tumor. The decision maker makes a final decision whether the input image is a cancerous tumor or not.

Figure 5.1 illustrates the model architecture. The architecture is modified from the VGG-16 network. The VGG convolutional neural network was proposed by the Visual Geometry Group (VGG) from Oxford's University in 2014. The number 16 represents the number of layers. It achieved an accuracy of 92.7% on ImageNet dataset, resulting top 5th place in Large Scale Visual Recognition Challenge 2014.

The model consists of 9 convolutional layers and 4 fully connected layers, which is smaller number of layers than the original VGG-16 network. The input goes through three VGG blocks and 4 fully connected layer blocks. Each VGG block is composed of three convolution blocks, dropout, and max pooling layer. Inside each convolution part, there are a convolutional layer, a batch normalization layer and an ELU activation function layer (See figure 5.1). Inside the fully connected block, a fully connected layer is followed by an ELU activation function layer. For the last fully connected layer block, a softmax layer follows a fully connected layer as it has to classify the output class.

First convolution block has 32 filters whose size is 3x3, and 1x1 for the last convolutional layer. All the convolution blocks have a stride of 1 and padding of 0. Except for the number of filters, other conditions stay the same for the remaining convolution blocks. As the input goes through each convolution block, the number filter size becomes double; the number of filters of the second block is 64 and that of the third block is 128.

5.4 Roulette

We are using the roulette when initializing the feature extractor and decision maker for each model. These models are initialized with a feature extractor and a decision maker which resulted in the highest AUC after running the model 200 times. All the three models share the same feature extractor to get the generalized feature extractor. To this end, the feature extractor for DSP is used for all the other body parts when deciding the best decision maker. The best feature extractor with the DSP among

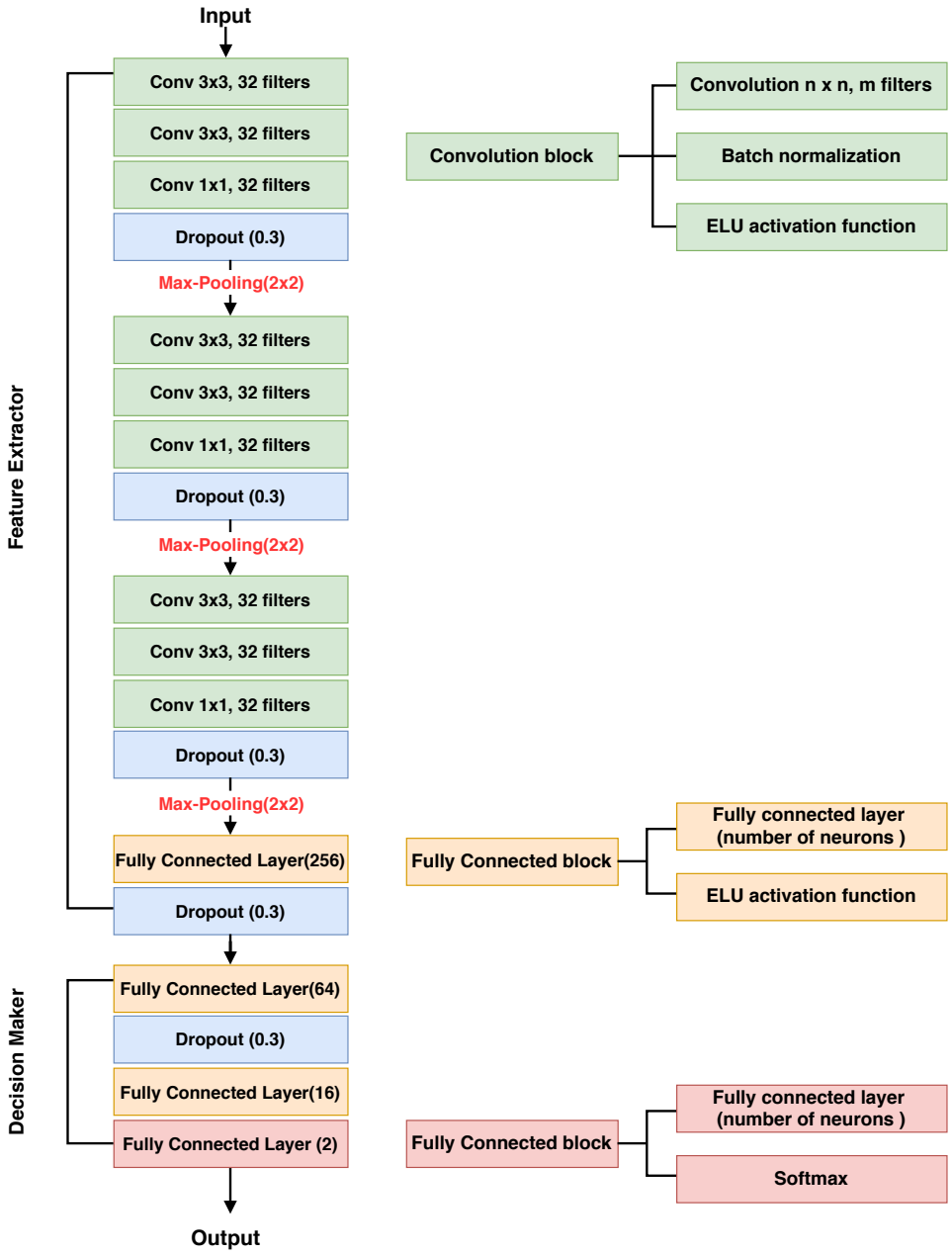

FIGURE 5.1: Model architecture. The left column indicates a model architecture with a description of each layer. The model is divided into two parts: feature extractor and decision maker. The right column shows a layer structure of a convolutional block and a fully connected block.

200 runs of the roulette is saved and used for other dataset's (DSB and DSL) decision maker initialization.

The roulette is validated by comparing between average AUC of 10 random initialization and the output of the roulette. After running 200 random runs, the final outcome is decided by picking the feature extractor and decision maker with the highest AUC. Ten random initialization is done and each AUC for each dataset is averaged for comparison. The AUC of the roulette had a higher accuracy than that of 10 random outputs.

AUC	Roulette Output	Average of 10 random runs
DSP	0.75887926	0.49394466
DSB	0.91688627	0.38305817
DSL	0.76782320	0.46263825

TABLE 5.2: Comparison between the roulette output and average AUC of 10 random runs. The "roulette output" column presents the highest AUC from 200 roulette runs. "The average of 10 random runs" column shows the average AUC of 10 random runs. For all the datasets, roulette output has a higher AUC. It was tested on the validation set.

5.5 Process Overview

All the datasets (DSP, DSB, DSL) that are used in this work follow the pre-processing step explained in Chapter 2. As our learning technique takes advantage of related tasks, we first need to verify if the tasks are related. In our case, the task is to diagnose if the patient has cancer or not (if the image has a malignant tumor or not). The detailed tasks for neural networks differ as the dataset contains three different body parts as each dataset have different features. We can see that the three tasks are related since the dataset is presumed to share common low-level features according to the experiments in Clement et al. [6] Each step of the learning process in cancer detection is schematized in Figure 5.2. And the description of each step is as follows:

- **Generalization**

  1. The best feature extractor (FE) and the best decision maker (DM) from the DSP are chosen from 200 random roulette initialization. After getting the best feature extractor (FE), it is attached to the decision maker of brain and lung datasets. After 200 runs, the best decision maker (DM) for both datasets are saved.
  2. The best feature extractor (FE) and decision maker (DM) for DSP as a whole model is trained.
  3. The same feature extractor (FE) and the decision maker (DM) from the roulette for DSB as a whole model is trained.
  4. The same feature extractor (FE) and the decision maker (DM) from the roulette for DSL as a whole model is trained.
  5. All three datasets are trained for 200 epochs while transferring the learned structure to each other.

6. After 200 epochs, each model that consists of the shared feature extractor (FE) and respective decision maker (DM) is tested on the test set.

- **Specialization**

1. The model from the generalization step is loaded for each dataset to train independently.
2. Each independent model is trained for 200 epochs and the best model which has the highest AUC on the validation set is saved and tested on each test set.

5.6 Training

For simplicity, training occurs in a fixed order (i.e. First DSP, second DSB then DSL). This sequence remains the same during the entire training. Three individual models3 consist of a shared feature extractor and a separate decision maker. The same feature extractor is trained and updated after every epoch. Each decision maker is also trained accordingly after each epoch. Several hyperparameter options (the learning rate and the dropout) are set differently for each model.

5.6.1 Evaluation Metric

The performance of the model is measured by the AUC metric. As explained in Chapter 3, AUC is a frequently used target metric in medical applications. For cancer detection tasks, it is important to see how confident the model is about its decision.

5.6.2 Visualization of a Model

To visualize performance of a model, we use Tensorboard, visualization a toolkit for tracking and visualizing metrics, visualizing the model graph, and displaying images and text. [31]

Throughout the training, seven evaluation metrics (loss, AUC, accuracy, precision, recall, f1score, and specificity) are computed and saved for both training and validation set respectively. After running all the epochs, the metrics of training and validation set are plotted in the same graph, so that both metrics can be compared and interpreted. The three models are then tested on the test set and their results are also recorded separately in Tensorboard. The full boards for the experiments described here can be found in chapter 6.

5.6.3 P-Hydra Learning Net

The model implemented in this work shares hidden layers with decision maker 1 (DM1), decision maker 2 (DM2), and decision maker 3 (DM3). The feature extractor is trained while being shared among all three tasks as common hidden layers. The model gets a medical image as an input. In the feature extractor, the hidden layers are learned together in parallel. The feature extractor is trained one after another in a fixed order (DSP, DSB, DSL). After the feature extractor outputs the feature, each feature is fed into each decision maker as an input. Unlike the feature extractor, decision maker is not shared among three tasks and trained individually just for their own target dataset. Finally, the decision maker decides the output class.

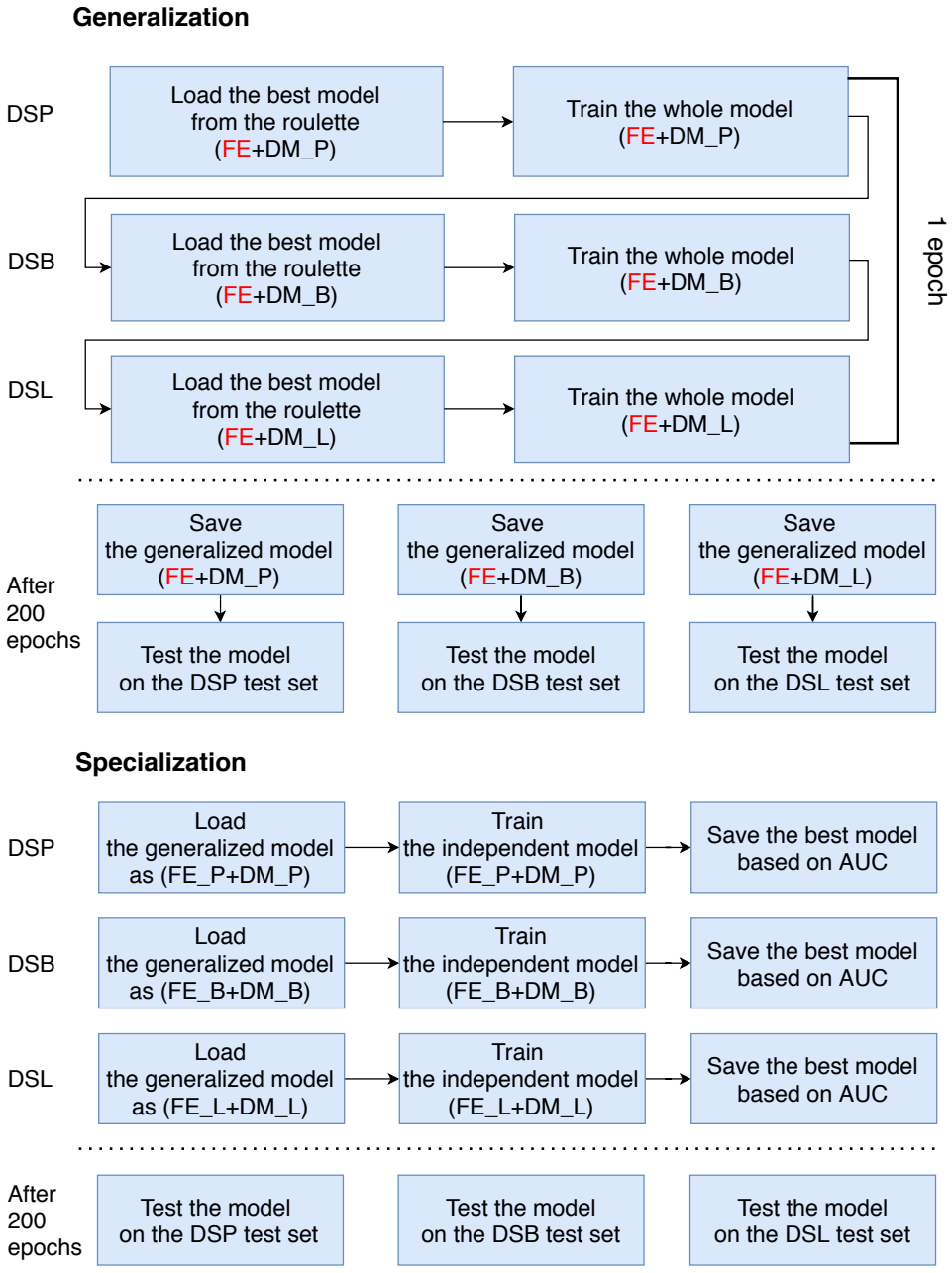

FIGURE 5.2: Process overview. Each model for each dataset is initialized from the roulette result. Feature extractor (FE) and decision maker (DM) for each dataset is combined as a whole model. Each model is trained for 200 epochs. After 200 epochs, each model is tested on the test set.

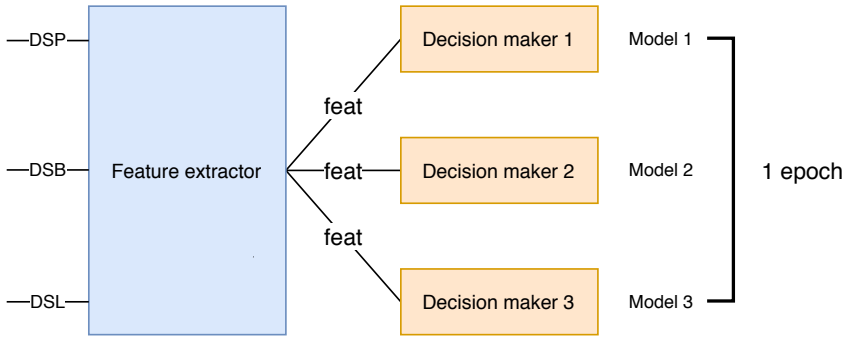

FIGURE 5.3: End-To-End model in training. For each model, End-To-End model consists of a shared feature extractor and a respective decision maker.

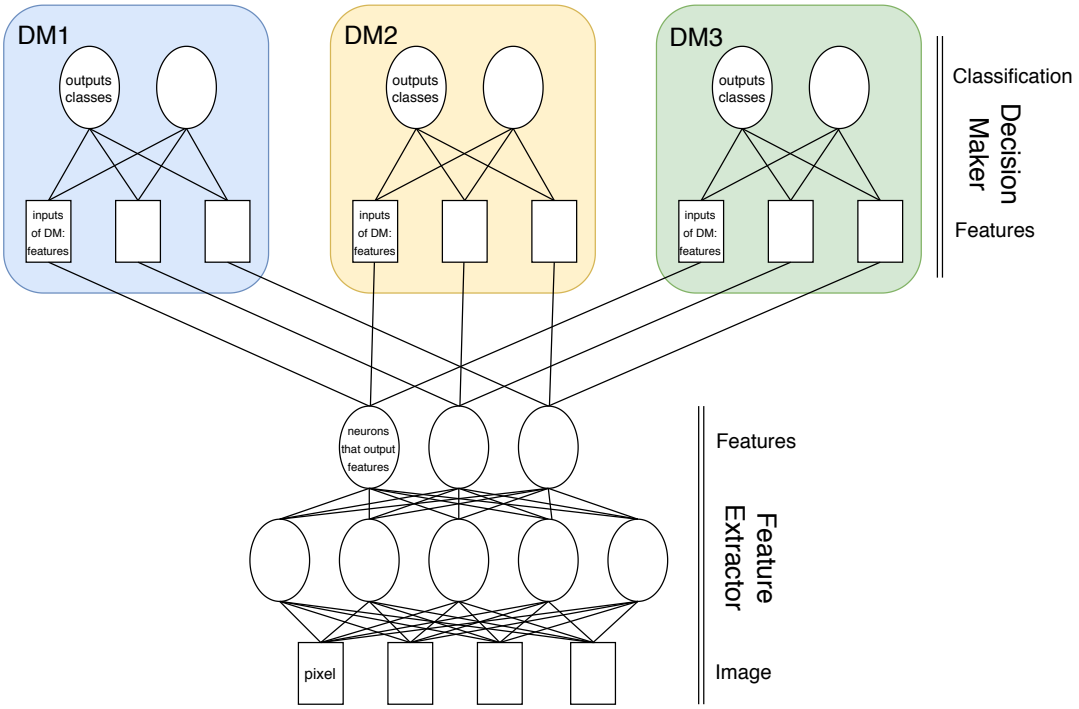

FIGURE 5.4: P-Hydra net. Input is a medical image. It goes through the feature extractor (FE) and useful features for every dataset are extracted. Then each decision maker (DM) is trained individually for the particular dataset. The FE is shared across models, while the DMs are distinct.

Chapter 6

Experimental Results

In this chapter, experimental results of P-Hydra learning are analyzed. This includes the explanation of hyperparameter option used for the experiment and corresponding training and validation graphs of seven metrics at each epoch and performance. The final results of each model (DSP, DSB, and DSL) are discussed.

6.1 Experiments

The experiment consists of two parts: generalization and specialization. After training the model with a shared feature extractor, we have a generalized baseline that can be used for specialization tasks. The model used for specialization does not share hidden layers, instead, it is an independent model that is specialized for one specific task. The generalization task is trained with 200 epochs and the specialization task for each dataset is trained with 200 epochs.

Several experiments are executed to tune the hyperparameters manually. Experiments with smaller epochs are first tested. With the small epochs experiment, a rough decision whether to run more epochs or not can be made. If the model shows a good learning process with a smaller epoch, it is trained with more epoch. Learning rates of 1e-5, 1e-6, 1e-7, and 1e-8, batch size of 64, 128, and 256, dropout probability of 0.2, 0.3, and 0.4 are tested for hyperparameter tuning. PyTorch is used to implement the model on conda environment. Adam optimization algorithm is used for updating the network weights.

6.1.1 Experimental Setup

The machine used for the experiments is an Nvidia Tesla V100. It took approximately 8 hours to run 400-epoch-experiment (200 epochs for generalization and 200 epochs for specialization). Final result with training and validation graphs are represented with the help of Tensorboard. Hyperparameter options of the final result are shown on the Table 6.1.

6.2 Results

First, a learning rate of 1e-8 is set for three datasets. Due to the small learning rate, the model fails to learn helpful features and metrics such as accuracy, recall, and F1score are stuck. During the hyperparameter tuning process, learning rate and dropout probability had the most impact on the performance of a model. As a result, learning rate of 1e-5, batch size of 64, and the same dropout probability except for DSB are applied for each model. All the models are trained 200 epochs for generalization and 200 epochs for specialization. Figure 6.1 to 6.3 illustrate training and

Dataset	Learning rate	Batch size	Dropout	Epochs_gen	Epochs_spec
DSP	0.00001	64	0.3	200	200
DSB	0.00001	64	0.2	200	200
DSL	0.00001	64	0.3	200	200

TABLE 6.1: Hyperparameter options for each dataset. DSP and DSL model had the same learning rate of 1e-5, batch size of 64, and dropout probability of 0.3. Notice that DSB has a different dropout probability of 0.2. Epochs_gen is the number of epochs trained for a generalization task. Epochs_spec is the number of epochs for a specialization task. Each model is trained 200 epochs for generalization and 200 epochs for specialization.

validation graphs created by Tensorboard. Table 6.2 to 6.4 show the final accuracy and AUC of each model.

Task (DSP)	Accuracy	AUC
Generalization	0.4711	0.6160
Specialization	0.5562	0.5793

TABLE 6.2: Accuracy and AUC of DSP after generalization and specialization.

Task (DSB)	Accuracy	AUC
Generalization	0.7851	0.8734
Specialization	0.8091	0.8402

TABLE 6.3: Accuracy and AUC of DSB after generalization and specialization.

6.2.1 Discussion

The final result of hyperparameter tuning is explained in this section with the description of an individual model with its corresponding training and validation graph. X-axis is the number of epochs. Y-axis represents each metric value of training and validation. Both generalization and specialization are presented in the same plot.

DSP

The feature extractor from DSP works as common hidden layers which will be shared among all the other datasets. First, the best feature extractor and the best decision maker from the roulette are initialized as a whole model. The best initialization from the roulette is decided by the best AUC on the validation set of DSP. After each epoch, only the feature extractor is shared with the next models (a model for DSB and DSL) so that the next models can benefit from what is learned in this step. For the next steps, the hidden layers are trained to learn the common features that all the other datasets have. Figure 6.1 describes Tensorboard graphs of seven different

Task (DSL)	Accuracy	AUC
Generalization	0.5065	0.5992
Specialization	0.5584	0.6037

TABLE 6.4: Accuracy and AUC of DSL after generalization and specialization.

metrics. Until 200 epochs, it shows a generalization task. After 200 epochs, specialization part which uses an independent model for DSP is trained for 200 epochs. Thanks to the roulette initialization, the model could converge quickly in the beginning. It has a slow but steady improvement. The generalization task achieved an accuracy of 0.4711 and AUC of 0.6160. The accuracy and AUC achieved after specialization are 0.5562 and 0.5793.

DSB

An end-to-end model that consists of the common feature extractor and the best decision maker from the roulette is initialized. The best decision maker is chosen by the best AUC on the DSB validation set. It is trained while sharing the common feature extractor for first 200 epochs.

The DSB is trained with the learning rate of 1e-5, which is the same number as the first dataset (DSP). Slightly small dropout probability of 0.2 is applied on this model. Figure 6.2 illustrates training and validation graphs of the DSB model. Thanks to learned features from DSP and DSL, a model could achieve a high accuracy and AUC from the beginning. As the epoch increases, accuracy and AUC also improve. After 200 epochs, where the specialization task begins, it is observed that the training and validation curves become closer. Using generalized features, the specialization task could have a higher start and the validation loss starts to decrease. At the end of the generalization task, the DSB model achieved AUC of 0.8734 which shows that the model could make use of all the learned features and generalize well on an unseen dataset.

DSL

Like DSB, the common feature extractor and the best decision maker from the roulette are initialized. The entire model is trained with a learning rate of 1e-5, a dropout probability of 0.3 and a batch size of 64.

Corresponding training and validation graphs are presented in Figure 6.3. Like DSP and DSL, the model converges in the first epochs and training accuracy and AUC could achieve high values thanks to shared hidden layers and large learning rate. After generalization task, the model could have a higher start, thanks to the learned knowledge from that task. The training and validation graph shows that the model is learning features and improving. As the specialization task begins at 200 epoch, accuracy and AUC start going down until approximately 250 epochs, since the feature extractor is adapting to only one dataset (DSL). The DSL model achieved accuracy of 0.5065 and AUC of 0.5992 after generalization. Both accuracy and AUC increased when trained on DSL only as 0.5584 and 0.6037.

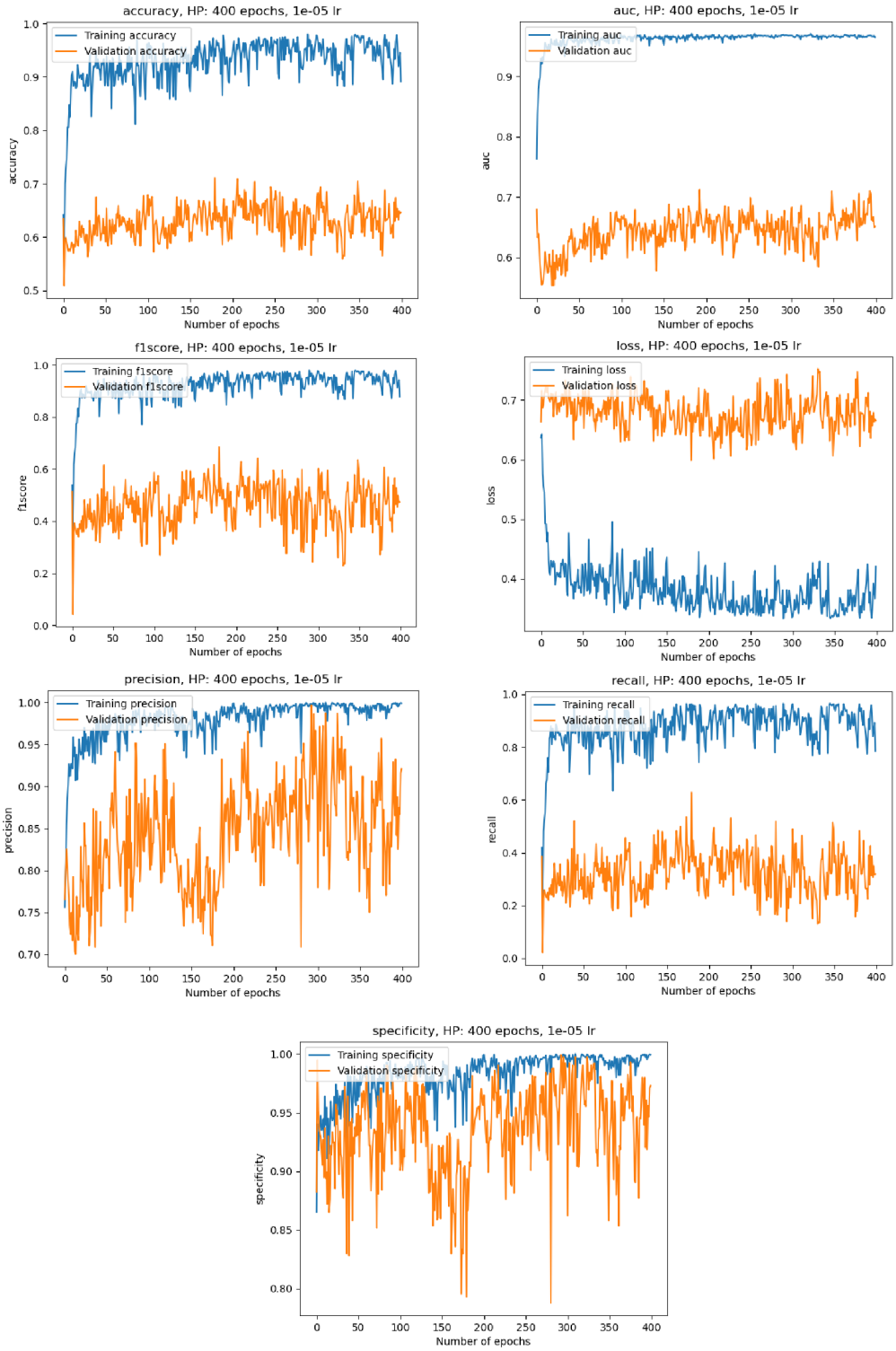

FIGURE 6.1: Final result. DSP with learning rate 1e-5, dropout of 0.3. Accuracy, AUC, f1score, loss, precision, recall, and specificity graphs are illustrated. The number of epochs are shown on Y-axis. X-axis represents a training and validation metric.

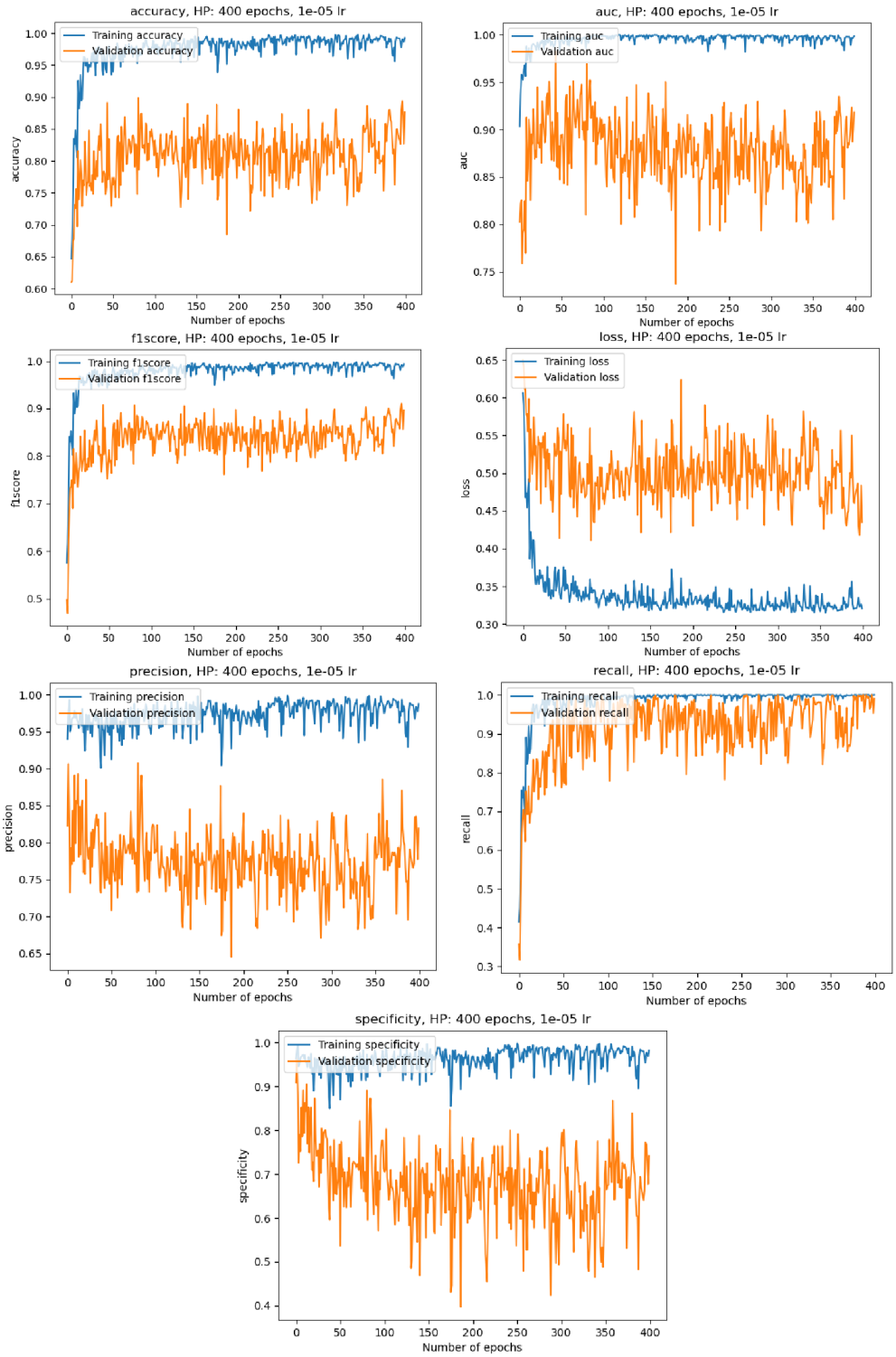

FIGURE 6.2: Final result. DSB with learning rate 1e-5, dropout of 0.2

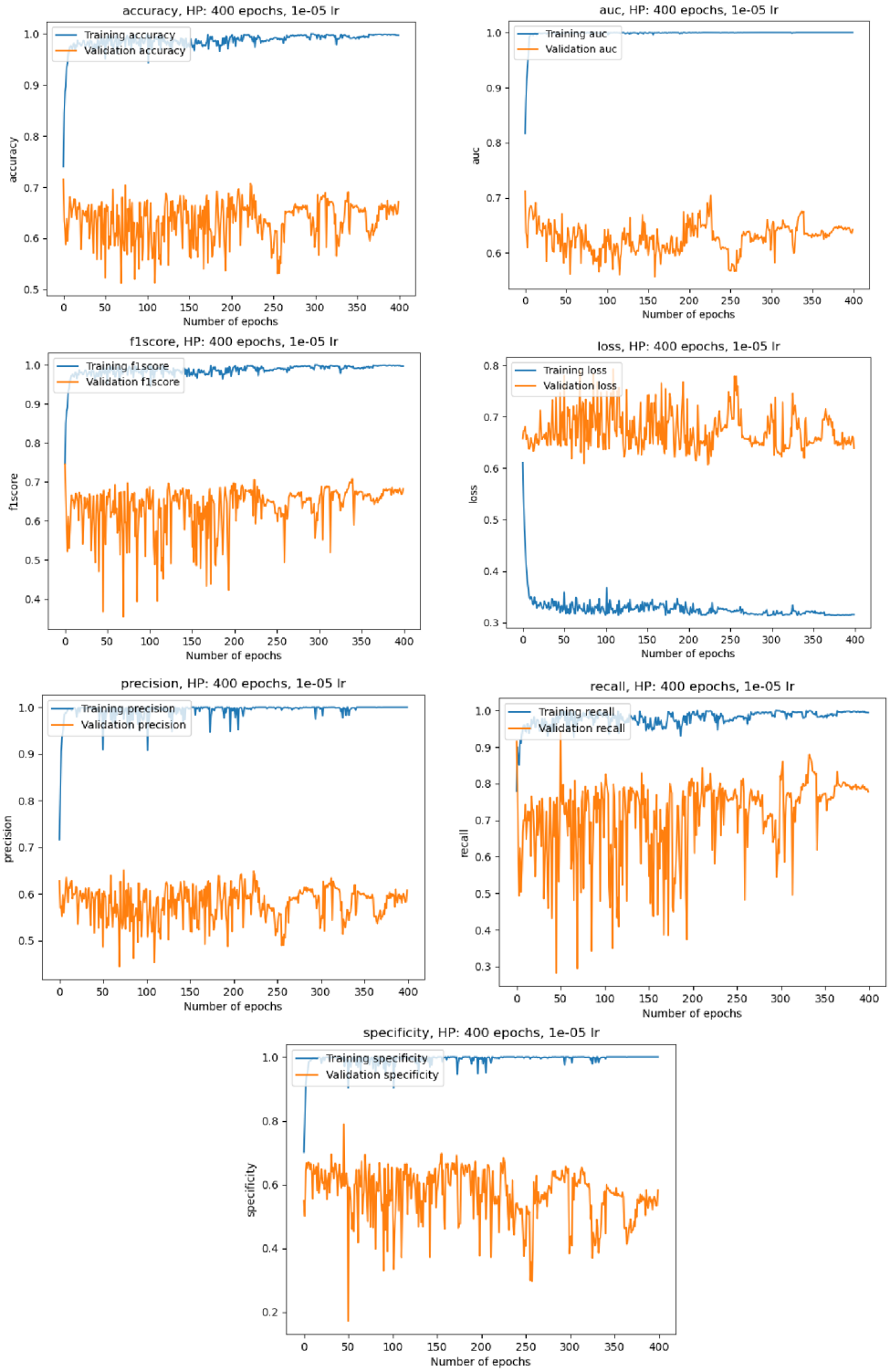

FIGURE 6.3: Final result. DSL with learning rate 1e-5, dropout of 0.3

Chapter 7

Conclusion

This work presented a novel approach to classify malignant tumors of three different body parts (prostate, brain, and lung). Deep learning with medical images (i.e. MRI, CT scans) usually suffers from lack of data. Two machine learning techniques, transfer learning and multitask learning, useful to overcome the data limitation problem are introduced. Transfer learning and multitask learning are similar in a sense that they both transfer the pre-trained information to another model. For transfer learning, the transferring process occurs after training is finished (sequential transfer), however, it occurs at the time of training for multitask learning (parallel transfer).

Our work is similar to transfer learning and multitask learning in that they both hand over the knowledge to a new model so that the new model can make use of the features acquired from the previous training. In contrast to our implementation, transfer learning cannot utilize information from related tasks since it is not available during training. With multitask learning, common hidden layers between related tasks make it possible to share what is learned for each task. The major difference between our work and multitask learning lies in specialization and generalization. Multitask learning focuses on specializing one specific main task by using auxiliary tasks on the same input whereas P-Hydra algorithm have three different inputs that are incompatible. As each decision maker is specialized on a completely different input, it is not possible to activate the three decision makers on the same input.

A state-of-the-art algorithm by Clement et al [6] is introduced for comparison. The work focused on developing a model that detects prostate cancer. However, our approach focuses on implementing a generalized feature extractor that can be used as a baseline for machine learning applications in cancer detection. Since there is a common problem in medical imaging fields: little availability of data, various methods had to be applied such as new learning techniques, and data augmentation. To train the feature extractor, three datasets (DSP, DSB, and DSL) were used. These datasets were assumed to have common low features, which means they could be used to train a generalized feature extractor. In addition to the generalized feature extractor, cancer detection models for three different body parts were developed.

Through the work, the learning pipeline is generated that can be used for other applications. One could produce a more generalized feature extractor with different hyperparameter settings, more data, or a different model architecture. The other option is using the feature extractor from our work as a baseline. Transfer learning could be applied for this case. For a cancerous tumor detection task using other datasets, the pre-trained feature extractor can be attached and then the only decision maker is trained for a different task. Using this sequential transfer approach can save time and effort for individuals who are implementing similar medical imaging tasks.

Several experiments with different learning rate, dropout probability, and batch size are conducted. Training with a very small learning rate (1e-8) resulted in a

constant accuracy problem. This is because the learning rate was too small to update weights properly (the update is too small). The same learning rate of 1e-5 was chosen for all the models and the same dropout probability of 0.3 was applied to except for DSB. For DSB, a smaller dropout probability of 0.2 was suitable to generalize better. The experiment shows that features from different body parts could help each other to generalize better. The generalized DSP, DSB, and DSL detection model had AUC of 62%, 87%, and 60% respectively. In the specialization task, where the model used a generalized model as a baseline, AUC achieved by each independent model are 58%, 84%, and 60%. Specialization tasks struggle as they try to adapt to the particular dataset. It was more difficult for the model to train on DSP and DSL.

7.1 Future Work

This work used MRI images and CT scans for three different body parts and produced a cancer detection model for each of them. As a result, a generalized feature extractor is produced that can be used for other medical applications. It can be used as a baseline and help a new model classify other medical datasets.

Although we picked another approach when it comes to distinguishing the main task and extra tasks. One can set extra tasks as a subset of the main task then train the model to perform a classification task (tumor, cancer detection) or even a segmentation task (location, coordinates of a tumor). For example, when the main task is to detect whether a patient has a cancerous tumor or not, then extra tasks can be the size of a tumor, the location of a tumor, the intensity of a tumor, etc. Another method is adding a less quantized task as an extra output.

The task of this work is dichotomous (It is a cancerous tumor or not). One can make the task less quantized by changing it to predict probability rather than a discrete label (i.e. True and False). Since it has smoother representation, this can help neural networks learn more easily. [28]

One can improve the model by training it with different hyperparameter options. This work trains the model for 200 epochs for both generalization and specialization. Applying different hyperparameters for each task, or using learning decay, which is a technique to change learning decay by certain factor, could improve the model.

Appendix A

Appendix

A.1 Script Options

Having script options makes it possible to set the dataset and tune the hyperparmeters and other choices for training. Table 3.4 shows all the possible commands when training the model. The hyperparameters that we can tune are batchsize, number of epochs, learning rate, and dropout.

Command	Description	Required	Type
trainingset	A list of training datasets folder path	True	Comma-separated strings
validationset	A list of validation datasets folder path	True	Comma-separated strings
testset	A list of test datasets folder path	True	Comma-separated strings
batchsize	Batch Size	True	int
nbepochs	Number of epochs	True	int
lr	A list of learning rate of each model	True	Comma-separated floats
lossfunction	Loss function name of choice [L1Loss, MSELoss, CrossEntropyLoss]	False default = CrossEntropyLoss	String
cudadevice	Device to run the model	False default = cuda	String
dropout	Dropout probability	False default = 0.2	Float
optimizedmetric	Metric of choice to measure the best performing model ['auc', 'accuracy', 'precision', 'recall', 'f1score', 'specificity']	False default = 'auc'	String
featuredirectory	A path to a directory to load the initialized feature extractor	True	String
dmdirectory	A list of folder path that contains each decision maker	True	Comma-separated strings
outputdirectory	A list of roots of the output directory used by Tensorboard and to save the models	True	Comma-separated strings

TABLE A.1: Script options

Bibliography

[1] *Promoting Cancer Early Diagnosis*. en. Library Catalog: www.who.int. URL: https://www.who.int/activities/improving-treatment-for-snakebite-patients (visited on 08/02/2020).

[2] *Clinical radiology UK workforce census report 2018 | The Royal College of Radiologists*. URL: https://www.rcr.ac.uk/publication/clinical-radiology-uk-workforce-census-report-2018 (visited on 08/05/2020).

[3] *The NHS does not have enough radiologists to keep patients safe, say three-in-four hospital imaging bosses | The Royal College of Radiologists*. URL: https://www.rcr.ac.uk/posts/nhs-does-not-have-enough-radiologists-keep-patients-safe-say-three-four-hospital-imaging (visited on 08/05/2020).

[4] Christopher J. D. Wallis, Masoom A. Haider, and Robert K. Nam. "Role of mpMRI of the prostate in screening for prostate cancer". In: *Translational Andrology and Urology* 6.3 (June 2017), pp. 464–471. ISSN: 2223-4691. DOI: 10.21037/tau.2017.04.31. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5503955/ (visited on 08/03/2020).

[5] *(PDF) Prostate Cancer Detection using Deep Convolutional Neural Networks*. en. DOI: 10.1038/s41598-019-55972-4. URL: https://www.researchgate.net/publication/338069773_Prostate_Cancer_Detection_using_Deep_Convolutional_Neural_Networks (visited on 09/30/2020).

[6] Julien Clement and Johan Jobin. *Prostate Cancer Classification: A Transfer Learning Approach to Integrate Information From Diverse Body Parts*. English. Mar. 2020.

[7] Chigozie Nwankpa et al. "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". In: *arXiv:1811.03378 [cs]* (Nov. 2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378 (visited on 10/15/2020).

[8] *ResearchGate*. URL: https://www.researchgate.net/publication/284579051_Fast_and_Accurate_Deep_Network_Learning_by_Exponential_Linear_Units_ELUs/link/5700e96208ae1408e15e59ce/download (visited on 11/11/2020).

[9] URL: https://www.deeplearningbook.org/contents/mlp.html (visited on 10/15/2020).

[10] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. ISSN: 1533-7928. URL: http://jmlr.org/papers/v15/srivastava14a.html (visited on 11/09/2020).

[11] Zachary Chase Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. "Thresholding Classifiers to Maximize F1 Score". In: *arXiv:1402.1892 [cs, stat]* (May 2014). arXiv: 1402.1892. URL: http://arxiv.org/abs/1402.1892 (visited on 09/17/2020).

[12] Sarang Narkhede. *Understanding AUC - ROC Curve*. en. May 2019. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (visited on 09/18/2020).

[13] Haibing Wu and Xiaodong Gu. "Max-Pooling Dropout for Regularization of Convolutional Neural Networks". en. In: *Neural Information Processing*. Ed. by Sabri Arik et al. Vol. 9489. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 46–54. ISBN: 978-3-319-26531-5 978-3-319-26532-2. DOI: 10.1007/978-3-319-26532-2_6. URL: http://link.springer.com/10.1007/978-3-319-26532-2_6 (visited on 08/31/2020).

[14] *Study finds AI matches humans in analyzing images*. en. Oct. 2019. URL: https://www.healthcareitnews.com/ai-powered-healthcare/study-finds-ai-matches-humans-analyzing-images (visited on 10/12/2020).

[15] Xiaoxuan Liu et al. "A comparison of deep learning performance against health-care professionals in detecting diseases from medical imaging: a systematic review and meta-analysis". English. In: *The Lancet Digital Health* 1.6 (Oct. 2019). Publisher: Elsevier, e271–e297. ISSN: 2589-7500. DOI: 10.1016/S2589-7500(19)30123-2. URL: https://www.thelancet.com/journals/landig/article/PIIS2589-7500(19)30123-2/abstract (visited on 10/12/2020).

[16] *Cancer*. en. URL: https://www.who.int/news-room/fact-sheets/detail/cancer (visited on 10/08/2020).

[17] Frank Gaillard. *MRI sequences (overview) | Radiology Reference Article | Radiopaedia.org*. en-US. URL: https://radiopaedia.org/articles/mri-sequences-overview (visited on 09/22/2020).

[18] *Medical Connections Ltd - Hounsfield Units*. en-gb. URL: http://www.medicalconnections.co.uk/ (visited on 11/12/2020).

[19] *SPIE-AAPM-NCI PROSTATEx Challenges - The Cancer Imaging Archive (TCIA) Public Access - Cancer Imaging Archive Wiki*. URL: https://wiki.cancerimagingarchive.net/display/Public/SPIE-AAPM-NCI+PROSTATEx+Challenges (visited on 09/28/2020).

[20] *Brain MRI Images for Brain Tumor Detection*. en. URL: https://kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection (visited on 09/28/2020).

[21] Samuel G. Armato et al. "LUNGx Challenge for computerized lung nodule classification". en. In: *Journal of Medical Imaging* 3.4 (Dec. 2016), p. 044506. ISSN: 2329-4302. DOI: 10.1117/1.JMI.3.4.044506. URL: http://medicalimaging.spiedigitallibrary.org/article.aspx?doi=10.1117/1.JMI.3.4.044506 (visited on 09/30/2020).

[22] *Prostate cancer statistics*. en. Aug. 2018. URL: https://www.wcrf.org/dietandcancer/cancer-trends/prostate-cancer-statistics (visited on 09/26/2020).

[23] Kh Kalan Farmanfarma et al. "Brain cancer in the world: an epidemiological review". en. In: (), p. 5.

[24] Priyansh Saxena et al. "Predictive modeling of brain tumor: A Deep learning approach". In: *arXiv:1911.02265 [cs, eess]* (Mar. 2020). arXiv: 1911.02265. URL: http://arxiv.org/abs/1911.02265 (visited on 09/28/2020).

[25] *Lung Cancer Fact Sheet*. en. URL: /lung-health-diseases/lung-disease-lookup/lung-cancer/resource-library/lung-cancer-fact-sheet (visited on 09/27/2020).

[26] Emine Cengil and Ahmet Çinar. *A Deep Learning Based Approach to Lung Cancer Identification*. Feb. 2019. DOI: 10.1109/IDAP.2018.8620723.

[27] Jason Brownlee. *A Gentle Introduction to Transfer Learning for Deep Learning*. en-US. Dec. 2017. URL: https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (visited on 11/04/2020).

[28] Rich Caruana. "Multitask Learning". en. In: *Machine Learning* 28.1 (July 1997), pp. 41–75. ISSN: 1573-0565. DOI: 10.1023/A:1007379606734. URL: https://doi.org/10.1023/A:1007379606734 (visited on 10/01/2020).

[29] S. C. Suddarth and Y. L. Kergosien. "Rule-injection hints as a means of improving network performance and learning time". en. In: *Neural Networks*. Ed. by Luis B. Almeida and Christian J. Wellekens. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1990, pp. 120–129. ISBN: 978-3-540-46939-1. DOI: 10.1007/3-540-52255-7_33.

[30] L. Holmstrom and P. Koistinen. "Using additive noise in back-propagation training". In: *IEEE Transactions on Neural Networks* 3.1 (Jan. 1992). Conference Name: IEEE Transactions on Neural Networks, pp. 24–38. ISSN: 1941-0093. DOI: 10.1109/72.105415.

[31] *TensorBoard*. en. URL: https://www.tensorflow.org/tensorboard (visited on 10/21/2020).