Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

National School of Engineering and Sciences

Graduate Program

Master of Science in Intelligent Systems

Thesis Proposal

# Early Detection and Diagnosis of Breast Cancer Lesions in Digital Mammograms using Deep Convolutional Networks

by

Erick Michael Cobos Tandazo
1184587

Tecnológico de Monterrey

Monterrey, N.L., Jun 25, 2015

**Instituto Tecnológico y de Estudios Superiores de Monterrey**

**Campus Monterrey**

**National School of Engineering and Sciences**

**Graduate Program**

The committee members, hereby, recommend that the master's thesis proposal presented by Erick Michael Cobos Tandazo be accepted to develop the thesis project as a partial requirement for the degree of **Master of Science**, with a major in:

**Intelligent Systems**

**Thesis Committee:**

Dr. Hugo Terashima Marín

Principal Advisor

Por definir

Committee Member

Por definir

Committee Member

Dr. Ramón Brena Pinero

Director of the Master's Program in Intelligent Systems

Jun 25, 2015

# Contents

**Abstract**

Breast cancer is one of the most common and deadliest cancer in women but early diagnosis can greatly increase its survival rate. Computer-aided detection (CADe) and diagnosis (CADx) assist doctors in the search of microcalcifications and masses, signs of early breast cancer. We plan to apply convolutional networks to mammograms, x-ray pictures of the breast, to automatically detect these lessions. Convolutional networks are a recent development in machine learning that learn the relevant features for classification from data; in contrast, traditional techniques use complex handcrafted features. A few studies have used convolutional networks for breast cancer detection but we plan to introduce newer features and carefully tune the architecture to produce improved results. Additionally, this will be the first approximation to use deep learning techniques as part of an ongoing project in the institution that aims to develop a CAD system for breast cancer. This thesis proposal is presented for approval to obtain the degree of Master of Science in Intelligent Systems.

## 1 Introduction

Accurately diagnosing breast cancer is hard for current computational systems; in this thesis, we use deep learning to improve their performance.

Breast cancer is caused by abnormal cells that grow out of control forming tumors and invading surrounding breast tissue. It has the highest incidence rate of any cancer in the United States, an estimated 14.1% of cancer diagnoses in 2015, and the third highest mortality accounting for 6.9% of all cancer-related deaths. Among women, it is the most commonly diagnosed cancer (28.6%) and has the highest death rate (14.5%) besides lung cancer [**?**]. The American Cancer Society recommends that women aged 45 or older should get mammograms, images of the breast which show signs of tumor formation, annually or biennially [**?**]. We consider two of the lessions that can be found on a mammogram: clustered microcalcifications, tiny deposits of calcium that could appear around cancerous tissue; and breast masses, more direct signs of the existence of a tumor although often benign.

We focus on using mammograms to automatically detect these lesions and predict the probability of breast cancer on the patient. Although manual examination of mammograms has a high sensitivity rate, automatic examination could be used by radiologists as a second informed opinion or as a help in deciding which regions should be further analysed. It could also be used where doctors are unavailable. With this motivation, the department created a project to design a computer-aided diagnosis system (CAD) for breast cancer. This thesis falls under the scope of this project as the first attempt to use deep learning for breast cancer diagnosis.

Traditional CAD systems for breast cancer work as a pipeline where each stage uses different computer vision and machine learning techniques. An standard pipeline will, for instance, preprocess the image, identify and segment the relevant parts of the picture, extract features from the segmented parts and train a classifier on the extracted features. Although some successful systems are built in this manner, they have a few disadvantages: each stage is a separate component and hence each of them needs to be improved to notably improve overall results, it is composed of dependent stages so that changes on one component affect the performance of other parts of the system, it uses complex image vision techniques that are difficult to handcraft to segment the images and extract features, it requires expert knowledge to be properly tuned, among others.

We plan to investigate the potential of convolutional networks to replace some if not all of the stages of traditional image processing systems. Convolutional networks [?, ?], a natural extension to feedforward neural networks, are a statistical learning classifier that uses raw images as input and learns the important features for the classification task as it is trained. Convolutional networks work well with minimally preprocessed images, can be trained to be rotational and translational invariant and perform segmentation, feature extraction and classification in one step. In our case, convolutional networks simplify classification potentially reducing it to a single component that we can train from labelled data and tune to obtain better results. Although convolutional networks have some drawbacks, they are the state-of-the-art technology for object recognition [?] and we believe it is worthy to experiment with them.

Researchers have used small convolutional networks to detect breast masses from normal tissue [?] and individual microcalcifications from noise in the image [?, ?]. In these experiments mammograms were preprocessed, enhanced and potential masses and microcalcifications were segmented and presented to the network for classification. We plan to use the available data and aggresively augment it using rotations, translations and reflections so that we could train bigger networks that would potentially learn more complex features. Furthermore, we plan to apply some of the most recent advances such as rectified linear unit activations, max-pooling, dropout, etc. to new tasks related to breast cancer detection where they have not yet been applied. For instance, the detection and diagnosis of masses and clustered microcalcifications.

We start our experiments by training a simple convolutional network to detect masses and clustered microcalcifications, later we train a more complex network architecture including some of the most recent advances and finally use the gathered knowledge to build an optimal convolutional network with tuned hyperparameter. As further experiments and depending on the available time we plan to pretrain a convolutional network with a different image database and fine-tune it using our database, use an all convolutional architecture, account for the problem of using an unbalanced data set and use an ensemble of networks. We intend to learn whether convolutional networks can automatically detect and diagnose breast cancer lesions, what are the advantages of using more data, a bigger architecture and tuned hyperparameters and whether we can achieve results similar to those of traditional systems.

This document offers an insight into the problem with traditional methods for image analysis in Section 2. It exposes the particular objectives and hypotheses of the thesis in Sections 3 and 4. Section 5 presents a comprehensive background of the scientific concepts used throughout the document and lastly a detailed methodology and work plan are shown in Sections 6 and 7.

## 2 Problem Statement and Motivation

Breast cancer is the most commonly diagnosed cancer in woman and its death rates are among the highest of any cancer. It is estimated that about 1 in 8 U.S. women will be diagnosed with breast cancer at some point in their lifetime. Early detection is key in reducing the number of deaths from breast cancer; detection in its earlier stage (*in situ*) increases the survival rate to virtually 100% [?].

With current technology, a high quality mammogram is "the most efective way to detect breast cancer early" [?]. Mammograms are used by radiologists to search for early signs of cancer such as tumors or microcalcifications. About 85% of breast cancers can be detected with a screening mammogram [?]. This high sensitivity is the product of careful examination of the mammograms by experienced radiologists. A computer-aided diagnosis tool (CAD) could automatically detect and diagnose these abnormalities saving the time and training needed by expert radiologists and avoiding any human error. Computer based approaches could also be used by radiologists as a help during the screening proccess or as a second informed opinion on a diagnosis.

CAD systems are based on image and classification techniques coming from Artificial Intelligence and Machine Learning. Traditional CAD tools for breast cancer diagnosis are composed of three steps: feature extraction, feature selection and classification. In the feature extraction phase, the system uses filters and image transformations to preprocess the mammogram and find geometric patterns which are used to produce a set of features for the image; expert knowledge is sometimes used in this phase. Feature selection or regularization is used to focus only on the important features for the classification task. Once a vector of features is obtained for each image, an standard binary classifier can be used to perform the final detection or diagnosis. These techniques have been used for many years and are standard in the industry [1].

Despite its widespread use and efficiency, systems based on traditional computer vision techniques have various limitations that should be addressed to further improve its performance:

- There is no standard way of preprocessing mammograms. Some techniques are commonly used but their performances can vary.
- It uses handcrafted features. The features extracted from the image are chosen beforehand (maybe designed with the help of experts) and special filters and image techniques are used to extract them.
- Segmentation and image feature extraction are error-prone and could greatly affect the classification results.
- It normally uses a small patch of the mammogram and makes a prediction on that patch but it does not consider the entire mammogram neither to make a prediction on the patient or to account for correlation between patches.
- To produce good results it requires knowledge in various fields such as radiology, oncology, image processing, computer vision, machine learning, etc.
- It is composed of many sequential steps. At each stage, there are many techniques from which the researcher can choose and many parameters which have to be estimated. This represents a cost in time and results as it is improbable that the optimal selection of techniques and parameters is achieved.
- As it is a complex system with different subsystems involved many other issues can arise such as non desired or unknown dependencies between subsystems, difficulty to localize errors, maintainability, etc.

[1]See [?] for an example of a CAD system developed in this institution.

- The techniques currently used are complex but the improvements achieved are not substantial. Much work is needed to make only incremental improvements and it is hard to know to which part of the system dedicate more resources.

This project will center around using Convolutional Networks, a recent development in computer vision, (see Section 5.4) to tackle some of these limitations, especifically automate preprocessing, feature extraction and segementation, use entire mammogram images and simplify the system pipeline by using a convolutional network as a replacement for many steps traditionally performed in succesion.

## 3 Objectives

The main goal of this work is to succesfully apply convolutional networks in digital mammograms to detect and diagnose breast cancer lessions and to compare our results with those obtained by other groups working in convolutional networks for breast cancer diagnosis.

Particularly, there are various subgoals which we expect to achieve as the project advances:

- Develop a working pipeline for processing the mammographic images from our database and training a convolutional network. Essentially, this tool could also be used for other image classification tasks.
- Use a simple convolutional network to perform detection and diagnosis and study these initial results to guide further research.
- Show the viability of convolutional networks for breast cancer diagnosis.
- Analyze the performance of convolutional networks reported on the literature.
- Use the improved convolutional network in the IRMA database.
- Generate results that could produce a conference or journal article.
- Use some alternative convolutional networks to improve our results.
- Propose new ideas and methods for future research in the topic.

Initial exploratory research has not yet been performed and some of these particular objectives may be modified as the project progresses. Furthermore, some new research avenues could be taken if they seem promising, for instance, using convolutional networks with digital tomosynthesis images (3-dimensional x-ray images of the breast).

## 4 Hypothesis

Although a considerable amount of work on breast cancer detection and diagnosis has been done in the institution, this project will be the first approximation to using convolutional networks for efficiently detecting and diagnosing breast cancer. Convolutional networks are widely used for object recognition tasks and have shown very good results [**?**, **?**, **?**]. They have a big research community and have become one of the preferred methods to perform image classification tasks.

Due to the exploratory nature of this work we are not truly certain of the results that will be obtained. Nevertheless, we have a well established idea of what to expect. Our hypothesis is that applying convolutional networks to mammographic images will produce similar or better results than those obtained using more traditional computer vision techniques. Additionally, we do not expect that a simple convolutional network will suffice to obtain competitive results; we will need a more refined convolutional network with well fitted parameters. Furthermore, we believe that implementing convolutional networks for breast cancer will not be very difficult as it has already been done by other groups (see Section 5.5) and there is plenty of software for it.

### 4.1 Research Questions

Some of the questions which will be answered in this work are:

- Can we improve the results reported by other groups using convolutional networks? Is training a convolutional network on mammographic images better than computing numeric features from the mammograms and training a simple classifier?
- Is deep learning feasible with the resources we have? Is our data and computational power sufficient? Is there any advantage to use GPU acceleration?
- Can we simplify the pipeline for breast cancer diagnosis? Can preprocessing be replaced by more layers on the same convolutional network? Could we automatically join results for small patches to generate results on the entire mammogram?
- What are the best parameters for our convolutional networks (number of layers, number of units, kernel sizes, regularization, activation functions, etc)? Is there a big improvement on refining the network and tuning parameters?
- Should we train a convolutional network for each type of breast lession or could we use a single one with multiple outputs?
- What are the advantages of using a deep versus a shallow convolutional network?
- Could we use a convolutional network trained on a different database (such as the ImageNet database) to obtain features for mammographic images and use these features for classification?
- Are convolutional networks a good option for future research?

## 5 Background

We offer an introduction to some of the essential concepts needed to understand the rest of this document. We start by discussing breast cancer and mammograms in Section 5.1, we explore some basic concepts about classification and evaluation metrics in Section 5.2, in Sections 5.3 and 5.4 we give a short introduction into Artificial Neural Networks and Convolutional Neural Networks, we present an overview of how convolutional networks have been used for breast cancer diagnosis in Section 5.5 and finally we offer some practical advice for deep learning in Section 5.6.

## 5.1 Breast Cancer

*Cancer* is an umbrella term for a group of diseases caused by abnormal cell growth in different parts of the body. The accumulation of extra cells usually forms a mass of tissue called a *tumor*. Tumors can be benign or malignant: *benign tumors* are noncancerous, lack the ability to invade surrounding tissue and will not regrow if removed from the body; malignant or *cancerous tumors* are harmful, can invade nearby organs and tissues (*invasive cancer*), can spread to other parts of the body (*metastasis*) and will sometimes regrow when removed [?].

*Breast cancer* forms in tissues of the breast. The two most common types of breast cancer are *ductal carcinoma* and *lobular carcinoma*, which start in the breast ducts and lobules, respectively (see Fig. 1). Breast cancer *incidence rate*, the number of new cases in a specified population during a year, is the highest of any cancer among American women. Its *mortality rate*, the number of deaths during a year, is also one of the highest of any cancer [?].

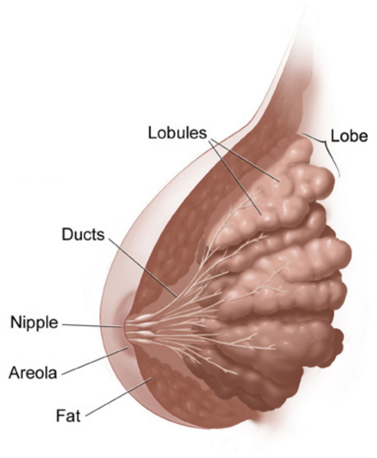

Figure 1: Anatomy of the female breast. Image courtesy of NCI.

The *cancer stage* depends on the size of the tumor and whether the cancer cells have spread to neighboring tissue or other parts of the body. It is expressed as a Roman numeral ranging from 0 through IV; stage I cancer is considered *early-stage breast cancer* and stage IV cancer is considered *advanced*. Stage 0 describes non-invasive breast cancers, also known as *carcinoma in situ*. Stage I, II and III describe invasive breast cancer, i.e., cancer has invaded normal surrounding breast tissue. Stage IV is used to describe metastatic cancer, i.e., it has spread beyond nearby tissue to other organs of the body.

### 5.1.1 Mammograms

A *mammogram* is an x-ray image of the breast. Radiologists use *screening mammograms* (normally composed of two mammograms of each breast) to check for breast cancer signs on women who lack symptoms of the disease. If an abnormality is found, a *diagnostic mammogram* is ordered, these are detailed x-ray pictures of the suspicious region [?]. A standard mammogram is shown in Fig. 2.

Having a screening mammogram in a regular basis is the most effective method for detecting breast cancer early; around 85% of breast cancers can be detected in a screening mammogram [?]. Nevertheless, screening mammograms have many limitations: a high false

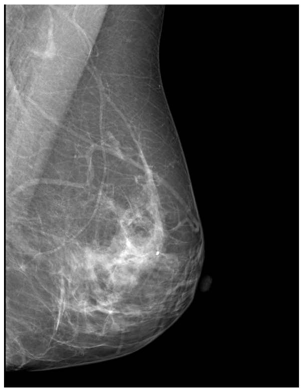

Figure 2: A standard mammogram.

positive rate, overtreatment in Stage 0 cancer, false negative results for women with high breast density, radiation exposure and physical and psychological discomfort [**?**].

Radiologists look primarily for microcalcifications and breast masses. *Microcalcifications* are tiny deposits of calcium in the breast tissue that can be a sign of early breast cancer if found in clusters with irregular layout and shapes. *Breast masses* or breast lumps are a variety of things: fluid-filled cysts, fatty tissues, fibric tissues, noncancerous or cancerous tumors, among others. A mass can be a sign of breast cancer if it has an irregular shape and poorly defined margins. See Fig. 3 for an example of possible signs of breast cancer. Radiologists will also consider the breast density of the patient when reading a mammogram given that high breast density is linked to a higher risk of breast cancer [**?**].

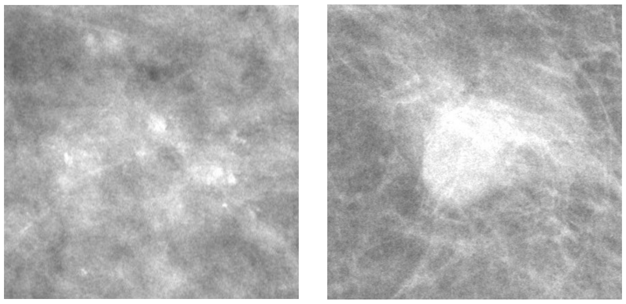

Figure 3: Signs of possible breast cancer in a mammogram. Left: A cluster of microcalcifications in an irregular layout. Right: A poorly defined breast mass.

Conventional mammography uses film to record x-ray images of the breast. *Digital mammography*, on the other hand, uses digital receptors to convert the x-rays into electrical signals and stores the image electronically. Digital mammograms offer a clearer picture of the breast and can be digitally manipulated and shared between health care providers. However, researchers still debate its effectiveness to identify breast cancer over film mammograms [**?**, **?**, **?**]. Digital mammography is steadily becoming the standard for breast cancer screening. Fig. 2 is, in fact, a digital mammogram.

*Digital tomosynthesis*, also called three-dimensional mammography, is a new technology that produces 3-dimensional x-ray images of the breast and is expected to improve the efficacy of regular 2-d mammograms. Studies comparing the two techniques have not yet been published [?].

We center on using mammograms, either digital or digitized from film, to detect microcalcifications and masses and predict the likelihood of breast cancer on the patient.

We wrote most of this section using information from the National Cancer Institute. We recommend to visit its website (`www.cancer.gov`) for further details.

## 5.2 Classification

*Machine learning* is the study of algorithms that build models of a population or a function of interest and estimate their parameters from data in order to make predictions or inferences. A machine learning expert knows how to choose the right model for the problem in hand (*model selection*), how to efficiently estimate its parameters from the available data (*learning* or *training phase*) and how to evaluate the trained model (*testing phase*).

Machine learning problems can be divided into three categories depending on the data used to train the model: *supervised learning*, where we learn a function $f(x)$ using a set of examples labelled with the correct output, for instance, learning a function that estimates the price of a house given its size and number of bedrooms from a dataset of houses labelled with their real value; *unsupervised learning*, where we look for relationships and structure in unlabelled data, for instance, given a dataset of potential customers find those who are likely to buy a car and *reinforcement learning*, where the only feedback received are rewards, for example, learning to play tetris from a dataset of world states and actions and where rewards are received sparsely every time points are earned (when lines dissapear). Supervised learning can be further divided in regression and classification. If the expected output is numerical, e.g., the price of a house, it is called *regression*, if the expected ouput is categorical, e.g., spam or no spam, it is called *classification*. We will focus on classification.

A *classifier* takes as input a vector of *features* $x \in \mathbb{R}^n$ from a problem instance and produces an *output* $h(x)$ predicting the class $y$ that instance belongs to, i.e., it concretely models the underlying function $f(x)$ as $h(x)$ ($h$ stands for hypothesis). *Binary classification*, when $y$ can only take two values e.g., cancer/no cancer, is the most common kind of classification and *multiclass classification*, when $y$ can take $K > 2$ different values, can be performed by using $K$ binary classifiers. Some classifiers, such as convolutional networks (defined in Section 5.4, output a *score vector* $h(x) \in \mathbb{R}^K$ where $h(x)_k$ is a measure of the probability that $x$ belongs to class $k$. Every classifier partitions the *feature space*, the $n$-dimensional space where features exist, into separate *decision regions*, regions of the space that are assigned the same predicted outcome; a *decision boundary* is the hypersurface that partitions the feature space. Classifiers are sometimes classified as *linear* or *nonlinear* according to the nature of the decision boundary they impose on the feature space. Logistic regression, for instance, is a linear classifier while an artificial neural network (with at least one hidden layer) is nonlinear.

The *loss function* $J(\theta)$ of a classifier measures the amount of error the classifier incurs in for a particular choice of parameters $\theta$. This function could be formulated in many ways. A *least-squares loss function* for a binary classifier (such as logistic regression) is presented in Equation 1

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

where $m$ is the number of training examples, $y \in \{0, 1\}$ is the real class of the example $x$ and $h_\theta(x) \in \mathbb{R}$ is the output of the classifier for input $x$ with parameters $\theta$, this represents the probability that $x$ belongs to the positive class 1. We introduce another (rather more complex) loss function in the next section.

A classifier is trained by choosing the parameters $\theta$ that minimize its loss function, hence, minimizing the expected error of the classifier on the training set. *Gradient descent* is a method used to estimate the parameters that minimize $J(\theta)$: at the start, it initializes parameters at random and iteratively updates each parameter using the gradient of the loss function until it converges to a minimum. Specifically, at each iteration it performs the update:

$$\theta = \theta - \alpha \nabla J(\theta) \tag{2}$$

where $\alpha$, called the *learning rate*, defines the step size. Gradient descent is guaranteed to converge to a global minimum if the loss function is convex, convexity of the loss function depends on the model $h(x)$.

To select the best model $h(x)$ for a particular problem or equivalently to select the best classifier for the problem each model is trained on a subset of the data set and later evaluated on a disjoint subset. In the validation set approach the data set is split into a training set (usually 70-90%) and a validation set, each model is trained using the training set and evaluated on the validation set and the model that shows the best performance is selected. *k-fold cross validation*, on the other hand, divides the data set in $k$ disjoint subsets (usually 5 or 10) and uses $k-1$ subsets to train the model and the remaining subset for evaluation, this process is repeated $k$ times for each model leaving out a different subset each time and the $k$ performance measures are averaged to obtain a final measure for the model. *Model hyperparameters*, settings that modify the underlying model or learning algorithm, are selected in the same way.

The model representation $h(x)$ needs to be chosen carefully. If we have an overly *flexible* model, i.e, $h(x)$ is a complex function with many parameters to be learned relative to the size of the training set, the classifier will probably *overfit* the data, this means that the parameters are fitted way too closely to the data and will pick up every small fluctuation and noise in the training set causing the trained classifier to produce almost perfect results on the training set but perform poorly on previously unseen examples. The opposite is also true, when $h(x)$ is very simple the classifier lacks the power to model the function of interest and we say that it *underfits* the data. This problem is sometimes referred as the *bias-variance tradeoff.* A high variance classifier is prone to overfitting, while a high bias classifier is prone to underfitting.

A popular way to avoid overfitting (and underfitting) is to use a flexible model trained with regularization. *Regularization* modifies the loss function to include a penalty to the complexity of the model, thus forcing the learning stage to choose parameters that minimize both the training error of the classifier and the complexity of the model. Equation 3 shows the least-squares loss function with *$l_2$-norm regularization*:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}||\theta||_2 \tag{3}$$

where $||\cdot||_2$ is the euclidean norm of a vector. In addition to reducing training error, minimizing the regularized loss function will shrinken the parameters $\theta$ hopefully setting some of them to zero, thus simplifying $h(x)$. The *regularization strength* $\lambda$ regulates the tradeoff between less training error and less regularization error. *$l_1$-norm regularization* or *lasso* is similar to $l_2$-norm regularization except that it shrinks the $l_1$-norm of $\theta$ instead of the $l_2$-norm.

To evaluate the performance of a classifier we use a separate set of examples (a test set) that should have not been used for training or validation. Classification accuracy is the standard performance measure in machine learning. *Accuracy* measures the proportion of test set examples correctly classified. Its compliment, *error rate*, measures the proportion of test set examples incorrectly classified. Accuracy, nonetheless, is inappropiate for unbalanced data sets, data sets that have many more examples of one class than the other e.g., cancer data sets are often unbalanced as most examples belong to the negative class (no cancer) than the positive class (cancer). For instance, a classifier that always predicts no cancer regardless of the input will show a high accuracy (equivalently a low error rate) even though it is a bad model for the problem.

A different set of metrics based on the confusion matrix of the classifier are used to evaluate its quality in unbalanced data sets. A *confusion matrix* is a matrix that summarizes the results of a classifier in the test set (see Table 1). *True positives* is the number of positive examples correctly predicted as positive. *False positives* is the number of negative examples incorrectly predicted as positive. True negatives and false negatives are defined in a similar fashion. Based on the confusion matrix we can compute some commonly used metrics:

| | | **Actual class** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted class** | Positive | True Positives (TP) | False Positives (FP) |
| | Negative | False Negatives (FN) | True Negatives (TN) |

Table 1: Confusion matrix for a binary classifier

$$Sensitivity \text{ or } Recall = \frac{TP}{TP + FN} \tag{4}$$

$$Specificity = \frac{TN}{FP + TN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

Sensitivity and specificity are usually preferred to present results in medical diagnosis meanwhile precision and recall are preferred in machine learning. *Sensitivity* measures the proportion of positive examples predicted as positive and *specificity* measures the proportion of negative examples predicted as negative. *Precision* is a measure of the proportion of examples predicted as positive that are actually positive. A good classifier will have both high sensitivity and high specificity or similarly, high precision and high recall. It is always useful to have a single metric to evaluate classifiers, for example, to choose between two models; we show two commonly used metrics in Equation 7 and 8.

$$F_1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7}$$

$$G\text{-}mean = \sqrt{Sensitivity \times Specificity} \tag{8}$$

The threshold of a classifier regulates the trade-off between sensitivity and specificity (or similarly precision and recall): a classifier with a low threshold is prone to classify examples as positive but will potentially produce more false positives thus having high sensitivity but low specificity and viceversa for higher thresholds. The *precision-recall curve* of a classifier is a line connecting the points produced by drawing its precision (on the y axis) against its recall (on the x axis) as its threshold varies. The *receiver operating characteristic curve* is defined similarly but plots sensitivity (also called true positive rate) against 1-specificity (also called false positive rate). See Figure 4 for an example. The *area under the precision-recall curve* PRAUC and the *area under the receiver operating characteristic curve* AUC summarize the performance of the classifier over all possible thresholds and can be used for model selection; they range from 0 to 1 with higher been better. As with the previous metrics, AUC is sometimes preferred for medical diagnosis while PRAUC is mostly used in machine learning. For unbalanced data sets, "using the classifiers produced by standard machine learning algorithms without adjusting the output threshold may well be a critical mistake" [**?**]. For this reason, it is preferable to use metrics that consider all possible thresholds (AUC or PRAUC) rather than those where only one threshold is considered ($F_1$ score or G-mean). However, the simpler metrics can be used as long as the threshold is obtained separately using a validation set.

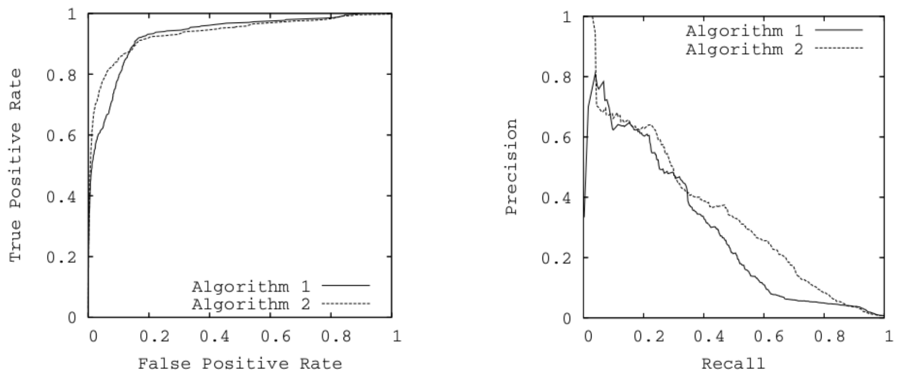

Figure 4: Sample receiver operating characteristic curve (left) and precision-recall curve (right). Each algorithm is evaluated on different thresholds and the points produced are used to obtain the curves. Image courtesy of [**?**]

In this thesis, we will generally present results for all these metrics (precision, recall or sensitivity, specificity, $F_1$ score, G-mean, AUC and PRAUC). The metric used when selecting a model influences its characteristics and behaviour, hence, one should put some consideration into choosing it; we favor the used of PRAUC over AUC (as well as $F_1$ score over G-mean) because they concentrate on predictions in the positive class (cancer) which is harder to predict and the class we are more interested in. Furthermore, PRAUC has been shown to have better properties than AUC in unbalanced data sets [**?**].

Finally, we point out that this section is meant to be a compendium of basic concepts in machine learning, practical machine learning involves many subtleties and implementation details not mentioned here. Notation and content in this section is mostly based on materials from Stanford's Machine Learning course[**?**].

## 5.3 Artificial Neural Networks

*Artificial neural networks* or simply *neural networks* are one of the most popular nonlinear classifiers used today. They were initially inspired by the way biological neurons process information coming from its dendrites and relaying it through its axon to neighboring neurons [**?**, **?**, **?**] but evolved to become more practical for nonlinear modelling albeit less biologically accurate [**?**]. We discuss here multilayer feedforward neural networks, the name should become obvious after a few paragraphs.

*Multilayer feedforward neural networks* are composed of $L$ layers of *neurons*, units of computation, each of which is fully connected to the next and previous layer (except for the first and last layer). The first layer, called the *input layer*, has $s^{(1)} = n$ units and receives the feature vector $x \in \mathbb{R}^n$ meanwhile the last layer or *output layer* has $s^{(L)} = K$ units corresponding to the $K$ possible classes. Every other layer is called a *hidden layer* (see Fig. 5 for an example). The neural network receives an input $x \in \mathbb{R}^n$, processes it layer by layer and outputs a vector $h_\Theta(x) \in \mathbb{R}^K$, where $h_\Theta(x)_k$ is the predicted (unnormalized log) probability that $x$ belongs to class $k$. Each unit performs a computation on the input from the units in the previous layer and transmits the result to the units in the next layer through its connections. Furthermore, each connection has a *weight* $w$ that is to be learned in the training phase, i.e, the weights are the parameters $\Theta$ of the model. A neural network is said to be *shallow* or *deep* according to its number of layers or *depth*.[2]

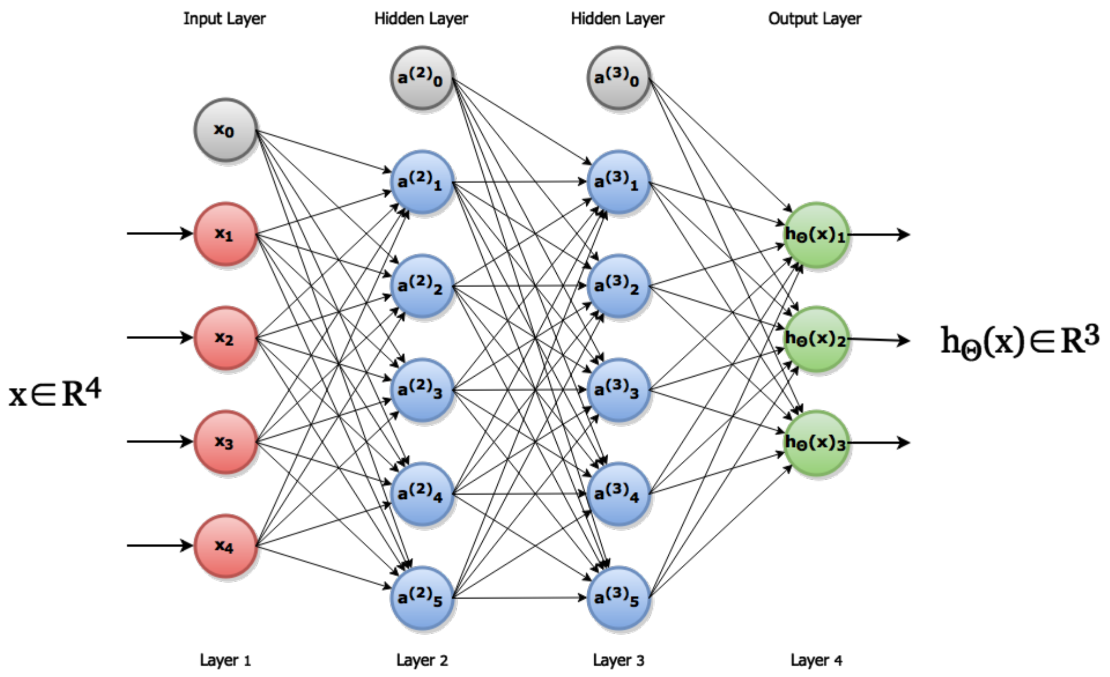

Figure 5: Small neural network example. Input layer with 4 units (red), two hidden layers of 5 units (blue) and output layer of 3 units (green). Bias units appear in gray. It approximates a function $h_\Theta(x) : \mathbb{R}^4 \to \mathbb{R}^3$, i.e., it classifies an input vector $x \in \mathbb{R}^4$ into 3 possible classes.

[2]Consensus on when a neural network becomes deep has not been reached [**?**]. We consider networks with two or more hidden layers to be deep.

A unit $i$ in layer $l$ computes a function of the form:

$$a_i^{(l)} = g\left(\sum_{j=0}^{s^{(l-1)}} \Theta_{ij}^{(l-1)} a_j^{(l-1)}\right) \text{ for } l = 2, \dots, L-1 \text{ and } i = 1, \dots, s^{(l)} \tag{9}$$

where $a_i^{(l)}$ is called the *activation* or output of unit $i$ in layer $l$; $g(\cdot)$ is an *activation function* (defined below); $s^{(l)}$ is the number of units in layer $l$; $a_0^{(u)} = 1$, for all $u = 1, \dots, L-1$ (bias units); $a_v^{(1)} = x_v$ for all $v = 1, \dots, n$ i.e, the activation of the input layer is the input $x$; $a_i^{(L)} = \sum_{j=0}^{s^{(L-1)}} \Theta_{ij}^{(L-1)} a_j^{(L-1)}$ for all $i = 1, \dots, s^{(L)}$ i.e, $g(\cdot)$ is omitted in the output layer and $\Theta^{(l)} \in \mathbb{R}^{s^{l+1} \times s^l}$ is the matrix of weights connecting layer $l$ to $l+1$. At each layer (except the output layer) we include a unit that always emits activation 1 ($a_0^{(1)} = 1$, $a_0^{(2)} = 1$, etc), these are called *bias units* [3]. The bias units are assumed to be included into each vector $a^{(l)}$, hence the sumation in Equation 9 starts at 0 and not 1. It may seem like a convoluted definition but it simply defines the activation of a given unit as the weighted linear combination of activations of the units in the previous layer passed through a nonlinear function $g(\cdot)$. Lastly, notice that $a^{(L)} \in \mathbb{R}^{s^{(L)}}$, the vector of activations in the last layer of the network, is equal to the predicted (unnormalized log) probabilities $h_\Theta(x) \in \mathbb{R}^K$, also called a *score vector*. The class with the highest score in $h_\Theta(x)$ is the predicted class for the example $x$. We could also exponentiate each of these probabilities and normalize them to obtain a distribution of probabilities over the possible classes $K$ ($p(x) \in [0..1]^K$). This improves interpretability and preserves the predicted classes.

The activation function $g(\cdot)$ is usually a *rectified linear unit* or *ReLU*:

$$g(z) = \max(0, z) \tag{10}$$

This is a nonlinear activation function very easy to implement in numerical computations, its derivative ($1_{z>0}$) can be calculated quickly and, unlike the sigmoid or tanh activation functions, does not suffer from vanishing and exploding gradients. Furthermore, it has been shown to greatly accelerate the convergence of gradient descent [**?**]. For these reasons it is currently the recommended activation function for deep neural networks [**?**].

Each unit in a neural network outputs a nonlinear activation $g(z)$ that in turn is received by units in the next layer, linearly recombined with the activation of other units and passed again through a nonlinear function; these operations are repeated until the input reaches the output layer. As a result, the function calculated by units in the output layer $h_\Theta(x)$ will be highly nonlinear on the original input $x$. This is the reason why neural networks can model functions that are highly nonlinear and why increasing the number of layers in a neural network increases the predictive power of the model. By the same token, it may be insightful to think of each unit in a neural network as a feature detector: units in the first hidden layer are trained to activate when simple features are found on the input, units on the second hidden layer activate when a combination of those simple features is present on the input and so on. Thus, the network will learn to detect the most relevant features for the classification task and as the number of units increases, it learns ever more complex features (if enough training data is provided).

[3] They are included for a technical detail: so that the activation function $g(w^T x + w_0)$ can shift in the x-axis changing its threshold to $-w_0$, which is learned by the neural network.

The *softmax* loss function for a multiclass neural network classifier is defined as:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{e^{h_\Theta(x^{(i)})_{y^{(i)}}}}{\sum_{j=1}^{K} e^{h_\Theta(x^{(i)})_j}}\right) \tag{11}$$

where $m$ is the number of examples in the training set, $h_\Theta(x)$ is the score vector, $K$ is the number of classes and $(x^{(i)}, y^{(i)})$ is the $i^{th}$ example. $J(\Theta)$ is differentiable with respect to $\Theta$ but non-convex, nonetheless, gradient descent usually converges to a good estimate of the network weights [?]. *Error backpropagation* [?, ?], an algorithm to calculate the derivatives with respect to $\Theta$ needed for gradient descent, computes the error terms in the output layer and backpropagates them layer by layer using the chain rule of derivatives. Deep neural networks are often susceptible to overfitting given the amount of parameters that need to be estimated. The simplest approach to overcome this problem is to use regularization. Regularization for neural networks is done by performing gradient descent on the regularized loss function presented in Equation 12

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{e^{h_\Theta(x^{(i)})_{y^{(i)}}}}{\sum_{j=1}^{K} e^{h_\Theta(x^{(i)})_j}}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s^{(l)}}\sum_{j=1}^{s^{(l+1)}}\left(\Theta_{ij}^{(l)}\right)^2 \tag{12}$$

*Dropout* [?] is another popular method to prevent overfitting. Each training iteration, dropout samples a different network architecture from the original network and updates only a subset of the values in $\Theta$; a unit (and its connections) is retained with probability $p$, usually 0.5, and gradient descent works on this sampled network. See Fig. 6 for an example. During testing all units are active but their activations are scaled by $p$ to match their expected output $(pa_i^{(l)} + (1-p)0)$. This can be interpreted as training many models (with shared weights) and averaging their results at test time.

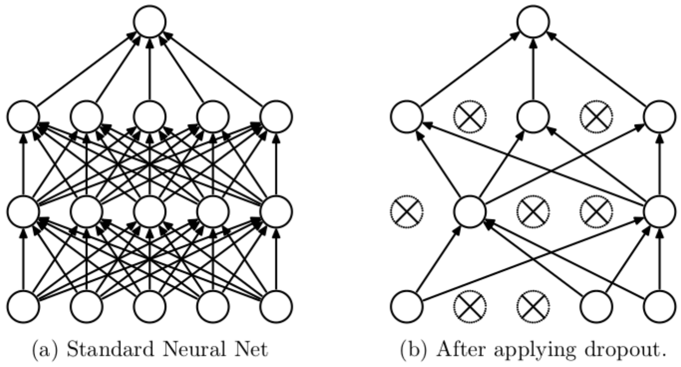

Figure 6: Dropout applied to a simple neural network. Crossed units were dropped. Image courtesy of [?].

## 5.4 Convolutional Networks

*Convolutional networks*, also *ConvNets* or *CNNs*, were first inspired by the way the human visual cortex proccesses information [?] but, as regular neural networks, have evolved to favor practical performance over biological accuracy. LeCun et al. used a convolutional network to achieve good classification performance on the MNIST data set of handwritten digits [**?**, **?**], the first successful application of modern convolutional networks. Recently, Krizhevsky et al. used them to achieve state-of-the-art performance on the ImageNet Large-Scale Visual Recognition Challenge [**?**], an image classification and object localization challenge with 1000 categories [**?**]. Thanks to various advances (maxpooling, ReLU activations, weight initialization, GPU training, efficient backpropagation, etc.), convolutional networks have become one of the most popular methods for image classification tasks and (along with recursive neural networks) an emblem of deep learning.

In this section we show the standard features and training of current convolutional networks. Section 5.6 gives some practical advice for choosing hyperparameters and training deep architectures. For an in depth review of convolutional networks, see [**?**]. For a complete overview of the history and state of deep learning, see [**?**].

Convolutional networks map raw image pixels to a score vector $h_\Theta(x) \in \mathbb{R}^K$ representing the distribution of (unnormalized log) probabilities over the $K$ classes. We could use a regular neural network (presented in Section 5.3) to do this but the amount of learnable parameters (the weights) becomes very big. For instance, a small color image of size $100 \times 100$ with 3 color channels (RGB) will require 30 000 units in the input layer and each unit in the second layer will therefore have 30 000 weights to learn. This is impractical not only because it will require a lot of data and training time but because the loss function has very many local minima and thus it is harder to optimize.

Convolutional networks reduce the number of connections between layers and the number of parameters to learn. Convolutional layers are *sparsely connected*, i.e., a unit is only connected to a small subset of the units in the previous layer. Furthermore, they are *locally connected*, i.e., units are connected considering their position on the original image. The architecture of a convolutional network also imposes *weight sharing* between units in the same layer, i.e., different units are forced to share the same weights (this determines filters and feature maps, defined below). *Pooling* is a subsampling mechanism that reduces the spatial scale and makes the computations invariant to local translation. All these features reduce computation and improve the classification performance of convolutional networks; they are produced by the way convolutional networks are defined, which we explain below.

Each layer is composed of a set of *feature maps*, 2-dimensional grids of unit activations ($\mathbb{R}^{h \times w}$), arranged into a 3-dimensional matrix ($\mathbb{R}^{h \times w \times d}$). We use the third dimension to put together all feature maps. One could unfold a 3-dimensional volume to form a single column of unit activations as in regular neural networks but seeing them as a volume makes the definitions easier. The input layer is a 3-dimensional matrix ($\mathbb{R}^{h \times w \times c}$) that holds the input image of size $w \times h$ with $c$ color channels ($c = 1$ for grayscale images or $c = 3$ for RGB). The output layer is a volume of size $R^{1 \times 1 \times K}$ where each feature map is just one activation ($R^{1 \times 1}$) representing the final score. The network receives an image $x$ as an input volume that is transformed layer by layer into new volumes (whose dimensions could be different from the previous one) until it reaches the output layer of size $h(x) = R^{1 \times 1 \times K}$. See Fig. 7 for an illustration. We describe the transformations next.

Four types of layers are commonly used: convolutional layer, ReLU layer, pooling layer and fully connected layer; all of which compute a differentiable function on its input and combine to form a convolutional network architecture.

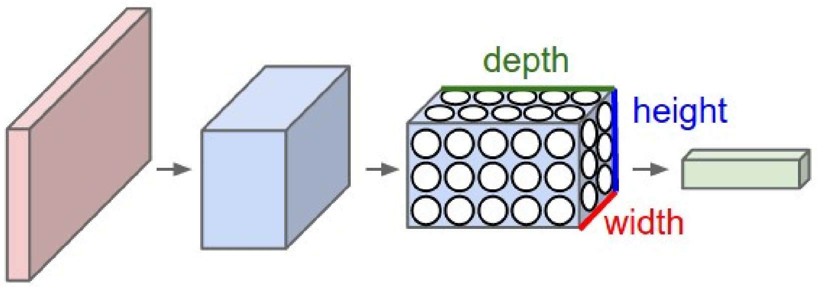

Figure 7: A simple representation of the transformations of the input computed by a convolutional network. Input layer is shown in pink, hidden layers are shown in blue and output layer is shown in green. The third layer has 5 feature maps of size $2 \times 3$. Notice that the width is listed first by convention. Image courtesy of [**?**].

**Convolutional layer** Convolutional layers are the heart of convolutional networks. They are build by learnable filters that are applied to the volume in the previous layer. A *filter* is a matrix of weights that has a small spatial size (width and height) but goes across all feature maps of the volume (the third dimension). For instance, a $3 \times 3$ filter to be applied in a volume with 10 feature maps will have 90 parameters ($\mathbb{R}^{3\times3\times10}$). See Figure 8 for an example. Each feature map in this layer is obtained by sliding a filter across the spatial dimensions (width and height) of the previous volume computing the dot product (a weighted sum) between the filter and the input [4]. All values in a single feature map are computed using the same filter. If we think of the feature map as a grid of units we can see that every unit is connected with only a small local subset of the units in the previous layer and that all units in the map share the same weights.

At each convolutional layer, many feature maps are computed (each with its own filter) and stacked together to form the volume in the layer. We can think of each filter as looking for an specific feature on the input and each feature map as the probabilities of that feature in each position.

We choose various hyperparameters for this layer: the filter size, the stride (the number of places to shift the filter at each step), the amount of zero padding around the image and the number of feature maps. These define the shape of the resulting volume; the first three are usually chosen to preserve the spatial size of the previous volume, the number of feature maps depends on the amount of features we want to learn.

**ReLU layer** This layer receives an input volume and performs an elementwise ReLU activation function to it, i.e, each value $z$ in the volume is passed through the nonlinearity $\max(0, z)$. It preserves the dimensions of the volume and has no learnable parameters, although the activation function may be considered a hyperparameter. Usually, a ReLU layer (or any other activation function) follows a convolutional layer, for this reason it is sometimes considered part of the convolutional layer. We separate them for clarity.

**Pooling layer** The pooling layer subsamples the volume on the spatial dimensions reducing the size of the feature maps but keeping the number fixed. Standard max pooling slides a fixed size windows (normally $2 \times 2$) along each feature map with stride 2 (it is, without overlapping)

[4]Each filter has also a bias term added to the sum.

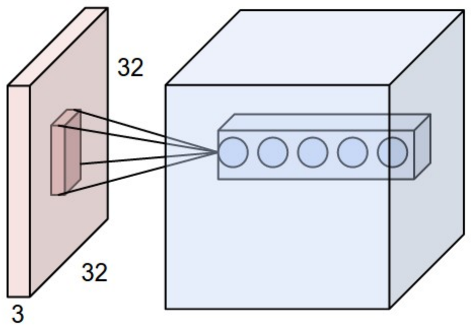

Figure 8: Example of a filter applied to a volume ($\mathbb{R}^{32\times32\times3}$) to obtain the values shown in the blue volume. The filter comprises all 3 feature maps of the input volume. We compute 5 feature maps as shown by the 5 units in the blue volume. For a complete convolution this filter will have to slide across the input volume. All units in the same feature map share the same filter but units in different feature maps do not, even though they can be connected to the same local region of the input. Image courtesy of [**?**].

and selects the maximum element on that space. This will reduce each dimension of the feature map by half, thus reducing the total number of activations by 75%, e.g., a $4 \times 4$ feature map gets subsampled to size $2 \times 2$ where each value is the maximum activation on each of the four quadrants of the original feature map. Contrary to convolution, subsampling is applied to each feature map separately. A popular variant of max pooling uses $3 \times 3$ windows with stride 2, allowing for some overlapping.

**Fully connected layer** We use one or more fully connected layers at the end of the network to compute the final score vector. Feature maps in this layer have size $1 \times 1$ resulting in a row volume or alternatively a row vector of values. Each feature map in this layer is fully connected to all units in the previous volume and outputs a dot product between the input and the connection weights, which are the parameters to be learned during training. The output layer of a convolutional network is always a fully connected layer with as many feature maps as classes. We interpret the scores of the output layer similar to those of regular neural networks as the (unnormalized log) probability of $x$ belonging to class $k$. Lastly, notice that a fully connected layer can be simulated by a convolutional layer with the same number of feature maps and filter size $w \times h$ where $w$ and $h$ are the spatial dimensions of the previos volume, i.e., filters that comprise the entire volume.

Convolutional layers (plus ReLUs) compute features on the input while pooling layers shrinken the volume before passing to the fully connected layers that act as a regular neural network classifier on the obtained features. The standard convolutional network architecture can be represented textually as:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

where `*N` indicates that the layers are repeated `N` times, `?` indicates that the layer is optional and `N,M,K >= 0`. We can use this template to construct ever more flexible models from a linear classifier `INPUT -> FC (N,M,K = 0)` to a regular neural network `INPUT -> [FC -> RELU]+ -> FC (N,M = 0, K > 0)` to a convolutional network `INPUT -> [[CONV ->`

`RELU]+ -> POOL?]+ -> [FC -> RELU]* -> FC` (`N,M > 0, K >= 0`). For instance, a typical deep convolutional network could be:

```
INPUT -> [[CONV -> RELU]*2 -> POOL]*3 -> [FC -> RELU]*2 -> FC
```

This network receives an input volume (the image) computes two sets of convolution plus ReLUs before pooling and repeats this pattern three times followed by fully connected layers plus ReLUs which are repeated twice and the output layer that reports the final classification scores. Although there is no standard way of counting the number of layers, we usually ignore the ReLU and pooling layers as they have no learnable parameters. Therefore, our example has 10 layers (21 in total), which is a good depth for big data sets. Practical recommendations on building architectures is offered in the Section 5.6.

Figure 9 shows an example of a convolutional network with its different kind of layers. The image is taken from a simulation accesible at `cs231n.stanford.edu`.

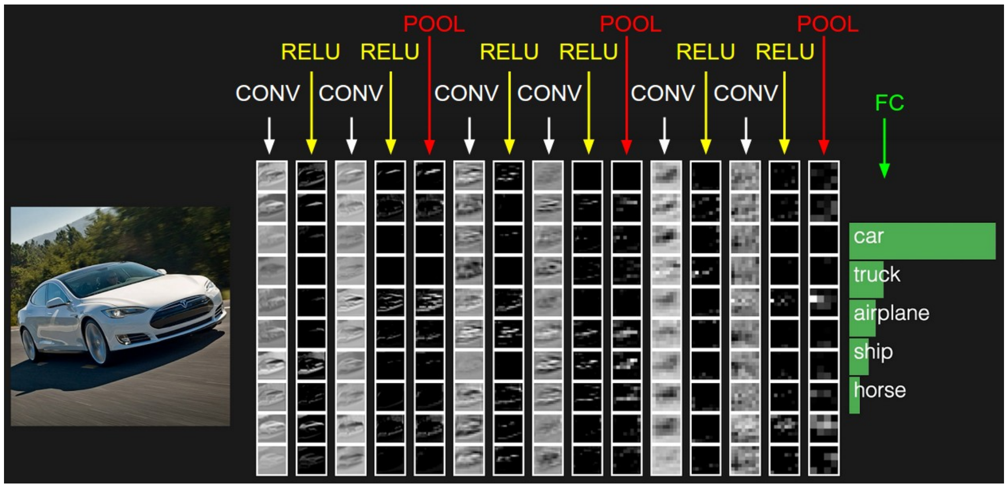

Figure 9: Example of a convolutional network with architecture `INPUT -> [[CONV -> RELU]*2 -> POOL]*3 -> FC`. The input image has size $32 \times 32$. Each hidden layer uses 10 feature maps (shown as columns). Although the spatial size of the feature maps looks constant in the image, each pooling layer reduces its dimensions by half (the feature maps of the final pooling layer have size $4 \times 4$). We show the final scores only for the five most probable classes. Image courtesy of [?].

Recently, simpler convolutional network architectures have emerged. The All Convolutional Net [?] is a network formed solely by convolutional layers: we replace pooling layers by convolutional layers with larger strides and fully connected layers as explained above. This greatly increases the number of parameters to be learn, therefore it is unsuitable for small data sets.

Converting the fully connected layers to convolutional layers has another advantage: we can use a convolutional network trained on small images to classify bigger images. By the way convolutional layers are defined when we use a bigger image as input the entire convolutional network will slide across the image and be applied to different portions of the image generating a score vector for each of them. Therefore, instead of having a single score for each class we will have an entire matrix of scores (for each position where the convolutional network was applied). Then, we can average over all scores per class to obtain a single score vector for the bigger image. Furthermore, we can control the stride of the convolution to choose how the convolutional network is slided across the big image. For instance, if we train a convolutional network with images of size $32 \times 32$ that via pooling get reduced to feature maps of size $4 \times 4$ in turn passed to the (converted) fully connected layers to obtain a score vector, then when using a $96 \times 96$ image as input to the same convolutional network it will get reduced to feature maps of size $12 \times 12$ and the fully connected layers will output a matrix of scores of size $9 \times 9$ (for each class), i.e, it slides the $4 \times 4$ fully connected layers across the $12 \times 12$ feature maps. Averaging each score matrix we obtain the final scores for the big image. We could have also set a stride of 4 in the first (converted) fully connected layer to get score matrices of size $3 \times 3$ for each 9 non-overlapping $32 \times 32$ partitions of the original image. It works exactly as if we applied the convolutional network to the original image at a stride of 32 but does all computations in just one pass. This way we can reuse a pretrained network to classify images of bigger size.

*Transfer learning* is a related method where we train a convolutional network on images from a specific domain and later use it to extract features on images from a different domain. It could also be used as a initialized network which is fine tuned with examples of the new domain.

The loss function for a multiclass convolutional neural network is similar to that for a regular neural network (Equation 12) except that now the convolutional network defines the vector score $h_\Theta(x)$.

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m} \log\left(\frac{e^{h_\Theta(x^{(i)})_{y^{(i)}}}}{\sum_{j=1}^{K} e^{h_\Theta(x^{(i)})_j}}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s^{(l)}}\sum_{j=1}^{s^{(l+1)}}\left(\Theta_{ij}^{(l)}\right)^2 \tag{13}$$

Furthermore, this loss function is still differentiable with respect to $\Theta$ and thus we can train the entire network via gradient descent. We use backpropagation to calculate the gradients of the loss function.

## 5.5 Convolutional Networks applied to Breast Cancer

Traditional CAD systems for breast cancer detection (CADe) and diagnosis (CADx) are normally composed of various succesive stages: preprocessing and image enhancement, segmentation of suspicious regions, feature extraction from the segemented image patches, optionally feature selection and region classification using machine learning methods. [**?**] offers a good review of the current state of traditional CAD systems. Here we focus on the previous application of convolutional networks for breast cancer detection and diagnosis in mammographic images.

As explained in Section 5.1 there are two main signs of breast cancer: breast masses and clustered microcalcifications, however, their presence does not necessarily imply cancer because they could be benign lesions. We refer as detection to the task of classifying a lesion as present or absent in the image no matter its malignancy, for instance, classifying an image patch as either clustered microcalcifications or normal tissue, while we talk of diagnosis when classifying a lesion as either benign or malignant [5].

**Overview** The only work using convolutional networks to detect masses was presented in [**?**]. Although the network produced competitive results, research focused more on feature-

[5]Images without lesions are considered benign.

based classifiers, such as regular neural networks, that produced better results using expert knowledge and advanced image techniques. Around the same time, [?] presented the first use of convolutional networks to detect individual microcalcifications. This work was followed up by [?] who looked to select an optimal convolutional network architecture for individual microcalcification detection. This optimized network was used in two CADe systems for clustered microcalcifications: one for film mammography [?] and one for digital mammography [?]. Both systems have a similar layout consisting of preprocessing, image enhancement, segmentation of potential microcalcifications, false positive reduction, regional clustering and false positive reduction of clusters. The convolutional network plays a relatively small role as an individual microcalcification classifier during the first false positive reduction stage. Lastly, recently an unpublished report [?] uses a convolutional network for the diagnosis of breast lesions obtaining average results.

In summary, convolutional networks have been sporadically used for breast cancer detection and diagnosis but have only played a minor role. Other than the most recent application, researchers have used very small networks (2 or 3 layers, <10K parameters) withouth any recent advancement (ReLUs, pooling layers, regularization, GPU computation) trained on very little data to classify image patches that were preselected using advanced image techniques. For instance, convolutional networks have only been used to detect individual microcalcifications but not clusters of microcalcifications and although they have been used to detect breast masses, using a larger network could probably improve those results. They have not been used for any type of breast cancer diagnosis either.

In the following sections we expand on the work mentioned above. We present a thorough summary of the most important articles organized by topic.

### 5.5.1 Detection of masses

To the best of the author knowledge, the first attempt to use convolutional networks for breast cancer is reported in "Detection of masses on mammograms using a convolution neural network" [?]. This 4-page article was later expanded in [?].

They used a small convolutional network (2 layers, $\sim$1K parameters) for the detection of masses [6]. Details of the best performing architecture can be found on Table ??. The data set consisted of 672 manually selected possible tumors from 168 digitized mammograms: out of which 168 were real tumors and 504 were not. Background reduction was performed on each image (using a rather convoluted method). The original images (size $256 \times 256$ pixels equivalent to a 2.56 $cm^2$ area) were downsampled via non-overlapping average pooling (filter size $16 \times 16$) to size $16 \times 16$; downsampling to $32 \times 32$ pixels via an $8 \times 8$ average pooling was also tried but gave similar results. Furthermore, the data was augmented by using 4 rotations (0°, 90°, 180° and 270°) on each original image and on each horizontally flipped image (8 in total per each training image) [7]. The network was trained via batch gradient descent plus momentum and per parameter adaptive learning rate. Two sets of experiments were performed: first, the $16 \times 16$ image patches (and their 8 rotations) were used for training producing 0.83 AUC on the best architecture, later these image patches were complemented with 2 $16 \times 16$ "texture-images" calculated using image techniques on the initial mass image

[6]They use the term mass to refer to tumors (either cancerous or non-cancerous) but not other kind of breast masses (cysts, fibroadenomas, fatty tissue, etc.). Thus, it actually detects tumors.

[7]The original article does not mention any data augmentation but it was probably performed given that they obtained the same results.

(a $16 \times 16 \times 3$ input volume) producing 0.87 AUC, 0.9 sensitivity and 0.69 specificity with the best network architecture. The authors showed that the network architecture was not as important for performance as providing the network with texture information. The texture features give back some of the information lost during the downsampling, which explains the improvement observed. The authors also acknowledge that the network architecture is far from optimal given its simplicity (one convolutional layer with three feature maps) and the incomplete hyperparameter tuning. A deeper network with more learnable parameters and a bigger input size could produce similar or better results without a need to include handcrafted texture features, which in theory could be learned by the network.

### 5.5.2 Detection of microcalcifications

The first use of convolutional networks to detect microcalcifications is reported in [**?**]. They performed various experiments on a small convolutional network (3 layers, ~5.4K parameters), details of the architecture are offered on Table **??**. The input size ($16 \times 16$), number of convolutional layers (2) and kernel size ($5 \times 5$) was obtained using a validation set, although only few options were explored: they tried input sizes of 8, 16 or 32, one or two hidden layers and kernel sizes of 2, 3, 5 or 13. A high sensitivity image technique was used to obtain a set of 2104 image patches ($16 \times 16$ pixels equivalent to an area of 1.7 $mm^2$) of potential microcalcifications from 68 digitized mammograms; of these, 265 were true microcalcifications and 1821 were "false subtle microcalcifications". Prior to training, a wavelet high-pass filtering technique was used to remove the background. Each image was flipped horizontally and 4 rotations for each the original and flipped image were used for training (0°, 90°, 180° and 270°). The network reached 0.89 AUC when identifying individual microcalcifications and 0.97 AUC for clustered microcalcifications; results obtained with a 30 fold cross validation. More than two individual microcalcifications detected on a 1 $cm^2$ area is considered a cluster detection, the predicted probability for the cluster is the average of the probabilities of all suspect patches inside the 1 $cm^2$ area [8] Other performance metrics were not explicitly reported. This article showed that deeper networks, background removal and data augmentation improved results. Together with the previous article, it also proved that simple convolutional networks can be used for breast cancer lesion detection.

A convolutional network with a similar architecture (3 layers, ~4.5K parameters) was presented by the same group in [**?**]. It detects microcalcifications from $16 \times 16$ image patches that were pre-selected and preprocessed using the same image techniques as above. For these experiments, nonetheless, they used 38 digitized mammograms and extracted 220 true microcalcifications and 1132 false ones that were randomly divided into a training and test set of roughly equal sizes. The network obtained a 0.9 AUC for individual microcalcifications and 0.97 AUC for clustered microcalcifications (also evaluated as in [**?**]). It showed that a convolutional network outperforms a regular neural network and a DYSTAL network in the clustered microcalcifications detection task when using raw pixels as input features.

[**?**] used simulated annealing (AUC as the objective function) to find the optimal number of feature maps and filter sizes for each of the two layers of a convolutional network used to detect clustered microcalcifications. The convolutional network was part of a bigger CAD system that first identifies the breast area and enhances it via a bandpass filter, later potential microcalcifications are segmented using adaptive thresholding methods, these suspected areas

[8]This method is not clearly explained either in this article or [**?**] so this interpretation may be incorrect. The way this evaluation is performed greatly affects the validity of the reported results.

are then filtered by a rule-based classifier that uses the contrast, size and signal to noise ratio (SNR) of the microcalcifications, the remaining image patches are fed to the convolutional network for final classification and lastly the individual microcalcifications are clustered to obtain the detected clustered microcalcifications. The convolutional network was trained on 1117 image patches ($16 \times 16$ pixels equivalent to 1.6 $mm^2$) obtained as explained above from 108 digitized mammograms without data augmentation. The best network found had 14 feature maps on the first convolutional layer with filter size $5 \times 5$ and 10 on the second layer with filter $7 \times 7$ (~7.6K parameters in total). The search ranges of each of these hyperparameters is not provided nor the specifics of the simulated annealing or the network training. The CAD with the optimized network reached 84.6% sensitivity at 0.7 false clusters detected per image. Other performance metrics were not provided. The authors show that optimizing the architecture of the convolutional network significatively improves the CAD performance. Nonetheless, given the small training set and incomplete hyperparameter search these result may be invalid.

[?] developed a complete CAD system for the detection of clustered microcalcifications in digital mammograms. This system is an evolution of that presented in [?] which was tailored to film mammograms. It consists of six stages: preprocessing, image enhancement via a box-rim filter, segmentation of potential microcalcifications using thresholding methods, classification of individual candidates using rule-based classifiers and a convolutional network, regional clustering via neighborhood growing and final classification of clusters using stepwise LDA feature selection and classification. We will focus on the convolutional network. It had the same architecture as before (shown in Table **??**) and was trained on around 500 $16 \times 16$ pixels image patches obtained from 48 digital mammograms: half of which were true microcalcifications while the other half were false microcalcifications. Its threshold was manually set to 0.4. The convolutional network reached 0.96 AUC for the detection of individual microcalcifications.

[?] compared the performance of both CAD systems in film mammograms and digital mammograms. They found that the performance in film mammograms is significatively better than in film mammograms which is intuitive given that digital mammograms have less noise and thus segmentation and feature extraction are sharper. Moreover, the convolutional network will be able to pick up more subtle details from the image without a need for further image enhancement.

### 5.5.3 Diagnosis of lesions

A recent unpublished report [?] uses a big convolutional network (8 layers, ~3.6M parameters) with recent features to diagnose lesions as benign or malignant. Clustered microcalcifications and masses are segmented from 8752 mammograms obtained from a public database (DDSM). Two to three square images ($64 \times 64$) are randomly sampled from each lesion and each of these is rotated 8 times (at 45° steps) producing 50K image lesions in total. No further preprocessing is performed. The convolutional network obtains 69.8% accuracy on the task. Other performance metrics are not provided. This is a project report for a course in Convolutional Networks [?] and may be incomplete or incorrect. We acknowledge it here for completeness.

| Article | Goal | Architecture | Volumes | Filters | # Params |
|---|---|---|---|---|---|
| [?] | Detect masses | `INPUT -> CONV -> SIGMOID -> FC -> SIGMOID` | $16 \times 16 \times 3$<br>$7 \times 7 \times 3$<br>$1 \times 1 \times 1$ | $10 \times 10$<br>$7 \times 7$ | 1047 |
| [?] | Detect individual microcalcifications | `INPUT -> [CONV -> SIGMOID]*2 -> FC -> SIGMOID` | $16 \times 16 \times 1$<br>$12 \times 12 \times 12$<br>$8 \times 8 \times 12$<br>$1 \times 1 \times 2$ | $5 \times 5$<br>$5 \times 5$<br>$8 \times 8$ | 5436 |
| [?] | Detect individual microcalcifications | `INPUT -> GAUSSIAN -> [CONV -> SIGMOID]*2 -> FC -> SIGMOID` | $16 \times 16 \times 1$<br>$12 \times 12 \times 10$<br>$8 \times 8 \times 10$<br>$1 \times 1 \times 2$ | $5 \times 5$<br>$5 \times 5$<br>$8 \times 8$ | 4530 |
| [?] | Detect individual microcalcifications | `INPUT -> [CONV -> SIGMOID]*2 -> FC -> SIGMOID` | $16 \times 16 \times 1$<br>$12 \times 12 \times 14$<br>$6 \times 6 \times 10$<br>$1 \times 1 \times 1$ | $5 \times 5$<br>$7 \times 7$<br>$6 \times 6$ | 7570 |
| [?] | Detect individual microcalcifications | `INPUT -> [CONV -> SIGMOID]*2 -> FC -> SIGMOID` | $16 \times 16 \times 1$<br>$12 \times 12 \times 14$<br>$6 \times 6 \times 10$<br>$1 \times 1 \times 1$ | $5 \times 5$<br>$7 \times 7$<br>$6 \times 6$ | 7570 |
| [?] | Diagnose lesions | `INPUT -> [CONV -> RELU]*2 -> POOL -> [CONV -> RELU -> POOL]*3 -> [FC -> RELU]*3 -> SOFTMAX` | $64 \times 64 \times 1$<br>$64 \times 64 \times 32$<br>$32 \times 32 \times 64$<br>$16 \times 16 \times 128$<br>$8 \times 8 \times 256$<br>$4 \times 4 \times 512$<br>$1 \times 1 \times 256$<br>$1 \times 1 \times 64$<br>$1 \times 1 \times 2$ | $3 \times 3$<br>$3 \times 3$<br>$3 \times 3$<br>$3 \times 3$<br>$3 \times 3$<br>$4 \times 4$<br>$1 \times 1$<br>$1 \times 1$ | 3.68M |

Table 2: Architectures of the different convolutional networks used for breast cancer detection and diagnosis.

## 5.6 Practical Deep Learning

In this section we collect some recommendations for constructing convolutional networks as well as efficiently training deep neural networks. These are intended to be specific to this project but most will also be useful in similar projects.

**Image preprocessing** Some standard processing for images.

- Images are cropped to contain only the relevant parts of the image, denoised, enhanced and optionally downsampled to maintain the input size fixed and manageable.
- Each image feature (the raw pixels) is zero centered by substracting its mean across all training images. Normalization scales the already zero-centered features to range from $[-1 \ldots 1]$ by dividing them by its standard deviations. Feature normalization is not striclty neccesary but still customary [?].
- The test data should not be used to calculate any statistic used to preprocess the training data. Furthermore, these same statistics (calculated from the training data) should be used when normalizing the test data [?].

**Convolutional network architecture** We offer some guidelines for designing convolutional network architectures and some standard values for various hyperparameters.

- It is always better to select a complex network architecture which is flexible enough to model the data and manage overfitting with regularization rather than an architecture which is not powerful enough to model the data [?, ?].
- Although, theoretically, neural networks with a single hidden layer are universal approximators provided they have enough units ($\mathcal{O}(2^n)$ where $n$ is the size of the input), in practice, deeper architectures produce better results using less units overall. This insight holds for convolutional networks [?].
- As a rule of thumb for big data sets, use 8-20 layers (not counting pooling or ReLU layers). For small data sets, use less layers or transfer learning. "You should use as big of a neural network as your computational budget allows, and use other regularization techniques to control overfitting." [?]
- Use the number of parameters rather than the number of layers or units as a measure of the architecture's complexity.
- Use 2-3 `CONV -> RELU` pairs before pooling (N above) [?]. Pooling is a destructive operation and having two convolutional layers together allows them to pick up more complex features.
- Use 1-5 `[CONV -> RELU]+ -> POOL` blocks (M above). This number depends on the complexity of the features expected in the data and the computational resources available. In a way, this regulates how much representational power will the architecture have. It also decides how much the volume is subsampled.

- Use less than 3 `FC -> RELU` pairs before the output layer (K above) [?]. When the volume arrives to the fully connected layers it has shrinken enough and using more fully connected layers risks overfitting.
- The number of feature maps per convolutional layer is set according to the expected number of features. This is similar to the number of units in a regular neural network. A common pattern is to start with a small amount of feature maps and increase them layer by layer [?].
- The number of feature maps per fully connected layer, or equivalently the number of units per fully connected layer, decreases from the number of units in the last convolutional layer (the number of units in each feature map times the number of feature maps) to the number of classes. For instance, having a convolutional network with two fully connected layers and 10 possible classes if the last convolutional layer produces a volume of size $8 \times 8 \times 512$ (8192 units), the first fully connected layer could have size $1 \times 1 \times 2048$ and the second (output) layer $1 \times 1 \times 10$.
- Use $3 \times 3$ filters with stride 1 and zero-padding 1 or $5 \times 5$ filters with stride 1 and zero-padding 2. This preserves the spatial dimensions of the volume and works better in practice [?]. When training on big images, the first convolutional layer uses bigger filters [?].
- Use $2 \times 2$ pooling with stride 2. Both this pooling and the overlapping version presented in Section 5.4 produce similar results. Pooling divides the spatial dimensions of the volume in half [?].
- Use square input images (width = height) with dimensions divisible by 2. The dimensions should be divisible by 2 at least as many times as the number of pooling layers in the network.
- Convert fully connected layers into convolutional layers.

**Hyperparameter search** We deal here with choosing hyperparameters other than those of the network architecture.

- Use a single sufficiently large validation set (20-30%) rather than cross validation [?]. For small data sets, cross validation can give better estimates and is preferred [?].
- Use random search rather than grid search. Random search draws each parameter from a value distribution rather than a set of predefined values. [?]
- Train each parameter combination for 1-2 epochs to narrow the search space. Later, train for more epochs on the refined ranges. Full convergence is not needed to make a decision on the hyperparameters [?].
- Hyperparameters related to the convolutional architecture, e.g., number of layers, number of feature maps, filter sizes, etc., are set manually (as explained above) rather than using a validation set.

- There are several hyperparameters to set: initial learning rate $\alpha$, learning rate decay schedule, regularization strength $\lambda$, momentum $\mu$, probability of keeping a unit active in dropout $p$, mini-batch size and type of image preprocessing.

- Theoretically we could fit all the hyperparameters using a validation set but in practice it is computationally unfeasible and could result in overfitting the hyperparameters to the validation data [?].

- Set $\alpha$, $\lambda$ and optionally the type of preprocessing using a validation set. Other hyperparameters can be set to a sensible default.

- The learning rate $\alpha$ is "the single most important hyperparameter and one should always make sure that it has been tuned" [?]. It ranges from $10^{-6}$ to $10^{0}$. Use a log scale to draw new values ($\alpha = 10^{unif(-6,0)}$ where $unif(a,b)$ is the continous uniform distribution) [?].

- The regularization strength $\lambda$ is usually data (and loss function) dependant. It ranges from $10^{-3}$ to $10^{4}$. Search in log scale ($\lambda = 10^{unif(-3,4)}$).

- If the best values for a hyperparameter are found in the limit of the range, explore further. [?].

- Use standard image enhancements. If there are no standard methods, use the validation set to choose from the options.

- Halve the learning rate every time the validation error stops improving. To obtain a fixed number of epochs, train the network (with the obtained hyperparameters) and observe when the validation error stops decreasing [?].

- Use $\mu = 0.9$. When using a validation set try values in $\{0.5, 0.9, 0.95, 0.99\}$ [?].

- Use 0.9-1 probability $p$ of retaining a unit in the input layer, 0.65-0.85 in the first 2-4 convolutional layers and 0.5 in the last convolutional layers and all fully connected layers [?]. Less dropout is used on the first layers because they have less parameters [?].

- Use mini-batch size of 64 or 32. A larger batch size requires more training time. It affects training time more than test performance [?].

**Training** Some general tips for efficiently training convolutional networks with million of parameters and very big data sets. Using these algorithms for small networks may be somewhat excessive but it will not hurt the performance.

- Randomize the order of the trainig examples before training. As we are using an stochastic estimator of the gradient this ensures the examples in each batch are sampled independently. Shuffling the examples after each epoch could also speed convergence [?].

- To estimate the number of examples needed to train a convolutional network divide the total number of learnable parameters by 25-100 (assuming some data augmentation). Some groups have been able to learn up to 40M parameters from as little as 60K training examples [?, ?].

- Weight initialization is very important for a proper convergence of the network. The current recommendation for ReLU units is to initialize each weight as a value drawn from a gaussian distribution $\mathcal{N}(\mu = 0, \sigma = \sqrt{2/n_{in}})$ where $n_{in}$ is the fan-in of the unit, i.e., the number of inputs to the unit. Specifically, each filter weight could be initialized as `w = randn()*sqrt(2/nIn)` where `randn()` returns a value drawn from a standard normal distribution and `nIn` is the number of connections to this filter (9 for a $3 \times 3$ filter, for example). Weights for units in the fully connected layer follow the same formula. Biases can be initialized likewise or to zero [?].

- Use mini-batches to compute the gradient. Using the entire training set to compute the gradient of the loss function takes a big amount of computation and points to the steepest descent direction locally but may not be the right direction if the update step is large. Using mini-batches allows us to make more updates, more frequently which results in faster convergence and better test results [?].

- Use Nesterov's Accelerated Gradient (NAG) to update the weights. It is a modified version of gradient descent which has shown to work slightly better for certain architectures [?]. Stochastic Gradient Descent with Momentum (SGD+Momentum) is also a viable option [?].

- Use dropout as a complement to $l_2$-norm regularization. Dropout usually improves results but it may slow network convergence [?].

- Store the network parameters regularly during training. Once per epoch should be enough but it depends on the number of parameters and size of the data. This allows you to come back to different versions of the network and select the one with the best overall validation/test error or one with some special characteristics [?].

- Stop the training process when the validation or test error has not improved since the last learning rate reduction. At this point gradient descent may not have converged but the validation error has and will start to increase (overfit) [?].

- Use the validation or test error to select the best parameters for the network from those stored [?].

- If you use the test set to refine a model, shuffle the entire data set and choose a diferent training and test set for the new model. Otherwise, you run the risk of overfitting to the test set [?].

**Sanity checks** Some simple checks to make sure the training is working properly.

- After weight initialization, the network should predict similar scores for each class (uniform probability) and have a loss function (without regularization) equal to $-\log(1/K)$. You can check this by running a test on a small set of examples. Adding regularization should increase the loss [?].

- If you implement back propagation manually or believe it may not be working properly you can run a gradient check. Gradient checks compare the analytic gradient produced by backpropagation with a numerical gradient produced by a finite difference approximation [?].

- Train the network with a very small subset of data (20 examples, for instance) and make sure it produces zero loss (without regularization). If it cannot overfit a tiny subset of examples the model is too simple [?].
- During training, the training loss should always decrease or only slightly increase. Otherwise, gradient descent may not be working properly either because of an implementation error or poorly tuned hyperparamters (high learning rate, low momentum) [?].
- Monitor the training and validation loss during training to identify overfitting and underfitting. Underfitting is characterized for a high training loss, overfitting is characterized for a big gap between training and validation (or test) loss [?].

**Data augmentation** One of the easiest ways to reduce overfitting in image data is to generate additional examples from the original data by applying some simple label-preserving transformations. Data augmentation allows the network to see different views of the same object thus enabling it to identify features that do not depend on the invariance introduced by the transformations. For instance, if we present it with images of a book on different rotations, we expect it to learn to identify a book no matter its position.

- There are many transformations one can apply: rotations, translations, horizontal and vertical reflections, crops (sample patches of the original image), zooms, etc. For color images, adding some noise to (jittering) the colors is also a valid transformation.
- Exploit the invariances you expect in the data set. For instance, galaxies are rotation invariant given that in space there is no up or down [?] but trees are not as it is rare to see an upside down tree.
- When combining different transformations in the same image be careful to preserve the original label. An overly modified image may lose its meaning.
- Most transformations are affine in the geometric plane and can be combined into a single one. If you plan to apply various transformations to the same image, applying a single affine transformation is faster and reduces information loss [?].
- Generate the augmented images during training. This saves storage and can be performed alongside the training [?].
- Data augmentation can also be used at test time by presenting the network with various versions of the same image and averaging its predictions [?].

**Unbalanced data** Having very few examples of one class compared to the rest is common in practice. We offer here some advice to deal with unbalanced classes using standard convolutional networks. We note that there is no accepted way to manage this problem.

- For a binary classifier, if the positive class is the rare class use PRAUC as a performance metric. If you are reporting the $F_1$ score, you should select an appropiate threshold using a validation set [?].
- If using PRAUC or selecting the threshold with a validation set is impractical, a simple adjustment is to divide the predicted probabilities by their corresponding class priors and to renormalize the values.

- For multiclass classification, use the macro-averaged $F_1$ score, an average of $F_1$ scores per class, with validated thresholds [?]. A multiclass PRAUC also exists but it is not as easy to interpret.
- As the threshold setting does not affect training it can be fitted independently of other hyperparameters once the network has already been trained. If fitting more than one, consider each one as a separate hyperparameter and use random search to find the best combination.
- One of the preferred methods to learn with unbalanced data sets is to use a modified loss function which gives a higher weight to errors in the rare class so that during training errors in the rare class will produce higher learning in the network parameters. Specific knowledge of the domain is required to estimate the cost of each class of error.
- Oversampling and undersampling, repeating the examples of the rare class or discarding some examples from the dominant class, are discouraged because they either not add information or throw away some of it.
- Replicating rare examples (oversampling) is useful when the examples are very scarce and the classifier simply does not have enough data to learn. This could be achieved by balancing the classes on each mini-batch via stratified sampling or by augmenting the rare class more than the dominant class during data augmentation.
- Data augmentation differs from data replication in that it only tries to enrich the data set with invariant images but actually leaves the proportion of classes unchanged.

**All convolutional networks** The research community has been moving towards discarding the pooling layers and using all convolutional networks. This can be interpreted as letting the network learn the pooling operation. We offer a couple of guidelines for implementing all convolutional networks.

- Replace each pooling layer with a convolutional layer with as many feature maps as in the previous layer and filter size $2 \times 2$ with stride 2 for normal pooling or $3 \times 3$ with stride 2 for overlapping pooling [?].
- For small data sets pooling layers also work as a regularizer because they reduce the number of learnable parameters and replacing them with convolutional layers may not be convenient [?].

**Transfer learning** When we have a small data set we could use a pretrained convolutional network either as a feature extractor for the new examples or to provide initializations for the new convolutional network, this is called transfer learning. We offer some tips for using a pretrained model specifically for mammographic images.

- Using a convolutional network pretrained in natural images, such as the ImageNet database, CIFAR-10, CIFAR-100, etc., may not work for mammographic images because features useful for one kind of classification are not very useful for the other. Nonetheless, given that features become more specific at higher layers, we could discard the higher layers of the network and use only the cropped network [?].

- Depending on the amount of data that we have we could: (1) add some fully connected layers on top of the pretrained network and train only these new layers, (2) add some convoutional and fully connected layers and train these new layers or (3) add convolutional and/or fully connected layers and train the entire network [?].
- When training on a pretrained model or fine-tuning use an smaller learning rate than when training a network from scratch. Using a small learning rate assures that we do not disturb very much the already good network parameters. [?].

**Software** A short description of four of the most popular packages for deep learning. They are pretty similar in capabilities and availability (open-source).

- Caffe [?]: Caffe is an already mature deep learning framework developed in C++/CUDA by the Berkeley Vision and Learning Center (BVLC) and community contributors. It offers a command line, Python and Matlab interface, reference models and tutorials and very fast code with easy GPU activation.
- Torch7 [?]: Torch is a scientific computing framework developed in C/Lua/CUDA at the IDIAP Research Institute. It offers n-dimensional arrays (tensors), automatic differentiation, a command line and Lua interface, GPU support and easy building of complex neural network architectures.
- Theano [?, ?]: Theano is a Python library developed in Python/CUDA at the University of Montreal. It is tightly integrated with NumPy, performs symbolic automatic differentiation and uses the GPU to efficiently evaluate mathematical expressions involving multi dimensional arrays.
- Cuda-ConvNet2 [?]: Cuda-ConvNet2 is a highly optimized convolutional network library developed in C++/CUDA by Alex Krizhevsky. It offers different off-the-shelf configurations for convolutional networks, a command line interface and multi-GPU training.

Lastly, we acknowledge that mammographic data is fairly different to that used in common object recognition tasks, for instance, labelling may not be as sharp or correct, images come in different sizes and ratios, image quality varies, the object to recognize may be very small in relation to the image, object localization may be a requirement, etc. and therefore some of the advice given above may prove counterproductive. When possible, design decisions should be based on the data and results obtained until that point.

## 6 Methodology

In order to achieve the proposed objectives and test our hypotheses we will need to carry out various tasks. We list them here in the order in which we plan to execute them:

1. **Literature review**

   A thorough review of the published work using the databases and resources available in the institution. By the end of this task, a complete theoretical background should be obtained and written. This will also help refine the scope of the project and the experiments to be conducted.

2. **Database preprocessing**

   We will ready the database images for the experiments; these implies joining different databases, obtaining the required features, preprocessing the images, assigning labels, etc.

3. **Software review**

   Once a clear idea of what are the possible experiments to be executed, we will need to find appropiate software to perform them. Software for database managing, preprocessing and implementation of different neural networks should be either located or developed.

4. **Exploratory experiments**

   We will train a standard convolutional networks with fixed parameters for the detection of microcalcifications and the detection of masses. We want to answer whether the convolutional network is powerful enough to learn the task in hand, whether we have enough data for the network to learn or more data augmentation is needed and wether the computational resources and parameter settings allow the network to learn in a timely fashion.

5. **Model selection**

   Using the insights from previous sections and the current literature on convolutional networks, we will select a network architecture along with novel features, preprocessing, training and regularization procedures. We aspire to find the best convolutional network configuration for mammogram classification.

6. **Further experiments**

   We will train the chosen convolutional network on our mammographic database. We will perform crossvalidation to adjust the most important network parameters and use regularization to avoid possible overfitting. We want to answer two research questions: is the performance of the convolutional network considerably improved by parameter tuning and, more importantly, is this a good performance?.

7. **Alternative convolutional networks**

   We will train a linear classifier, probably rectified linear units, on the features obtained from a convolutional network trained on the ImageNet database, i.e., we will use an already trained convolutional network instead of one trained specifically in mammograms. We will also use an all convolutional network of a size relative to the best arhcitecture up to that point. We want to answer two questions: can we use an already trained convolutional network to classify mammograms and do using an all convolutional network affects significatively the results?

8. **Gathering results**

   Produce results on the test set and elaborate figures and tables. This could be obtained directly from software output or from further program executions.

9. **Reporting results**

Write the thesis and any article or technical guide which may result from this work. Both this and the previous step will be performed along the execution of the project, hopefully benefiting from the supervisors' feedback.

Finally, we would like to note that this is an idealized workflow and some changes may occur due to time limitations or lack of resources. In the unlikely case that the work is finished before the project deadline, we will either reiterate on model selection, experiments and results gathering and reporting or look into digital tomosynthesis, network ensembles or evolving convolutional networks.

## 7 Work Plan

We present here the expected work plan for this master's thesis. A description of the activities can be found in Section 6

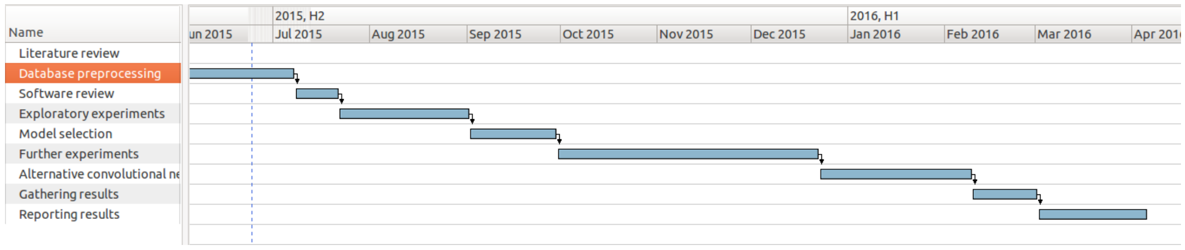

Figure 10: Thesis work plan.

Literature and software review and the preprocessing of the image database are expected to be done before the end of the summer term (late July). The first experiments on simple convolutional networks and applicability to breast cancer are expected for the fall semester (late October to early November). At this point, these results should be reported in the thesis and hopefully a scientific article. Later, the final architecture and methods will be selected and the final experiments run. The thesis document should be delivered by the start of March. During this month, if the new results are valuable, we expect to write a conference or journal article to share our results with the community.