Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Ταξινόμηση μαστογραφιών για καρκίνο του μαστού με τη χρήση στοιβαγμένων αποθορυβοποιητικών αυτοκωδικοποιητών** **Breast cancer classification of mammographies using stacked denoising autoencoders**
Ονοματεπώνυμο Φοιτητή	**Σπυριδούλα Τζιμογιάννη**
Πατρώνυμο	**Ιωάννης**
Αριθμός Μητρώου	**ΜΠΠΛ/ 12055**
Επιβλέποντες	**Γεώργιος Τσιχριντζής, Καθηγητής** **Διονύσιος Σωτηρόπουλος, Δρ.**

Ημερομηνία Παράδοσης **Φεβρουάριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα

ABSTRACT

Breast cancer is one of the most common cancers among women, and has also become a major cause of death. Medical image analysis is one of the less studied and challenging areas of computer vision and artificial intelligence in general. When we apply machine learning, obtaining reliable results is much more difficult for datasets with abnormalities, such as a dataset of mammographies, than for common applications, since the tissue types and the shape of the organs vary widely from person to person. However, with the use of deep learning, and, specifically, stacked denoising autoencoders, it is possible, with the appropriate tools and parameterization, to have promising, real-world results, that can successfully lead to the correct classification of medical images, hence providing great support to the medical for the early detection of breast cancer.

ΠΕΡΙΛΗΨΗ

Ο καρκίνος του μαστού αποτελεί έναν από τους πιο κοινούς τύπους καρκίνου για τις γυναίκες, και έχει γίνει, επίσης, κύρια αιτία θανάτου. Η ανάλυση των ιατρικών εικόνων είναι ένας από τους λιγότερο μελετημένους και πιο απαιτητικούς τομείς της υπολογιστικής όρασης και, γενικότερα, της τεχνητής νοημοσύνης. Όταν εφαρμόζουμε τη μηχανική μάθηση, η απόκτηση αξιόπιστων αποτελεσμάτων είναι πολύ πιο δύσκολη για σετ δεδομένων με ανωμαλίες, όπως σετ μαστογραφιών, από ότι για πιο κοινές εφαρμογές, μια και οι τύποι των ιστών και το σχήμα των οργάνων ποικίλλουν αρκετά από άτομο σε άτομο. Παρόλα αυτά, με τη χρήση της βαθιάς μάθησης, και, κυρίως, των στοιβαγμένων αποθορυβοποιητικών αυτοκωδικοποιητών, είναι πιθανό, με τα κατάλληλα εργαλεία και την κατάλληλη παραμετροποίηση, να έχουμε πολλά υποσχόμενα και πραγματικά αποτελέσματα, που μπορούν επιτυχώς να οδηγήσουν στη σωστή ταξινόμηση των ιατρικών εικόνων, παρέχοντας, έτσι, μεγάλη στήριξη στο ιατρικό προσωπικό στην προσπάθεια της έγκαιρης ανίχνευσης του καρκίνου του μαστού.

Contents

INTRODUCTION

Currently, mammography is one of the most effective means in early detection of breast cancer. Early detection of breast cancer leads to a reduction of breast cancer mortality by approximately 25%, since with the help of mammograms it is possible to capture potentially dangerous abnormalities, before the appearance of symptoms on the patient.

However, the interpretation of a mammogram is not an easy job. The experience of the radiologist plays the most important role, since the assessment is done with the comparison between the left and right mammogram, or with the previous exams.

So, in such an abnormal dataset, computer-aided diagnosis (CAD) might have promising, real-world results. Nowadays, image processing and image analysis are used widely to assist radiologists in detecting abnormalities and tumors in mammography images. In order to improve the results, we classify the masses as benign or malignant. In this experimental study we investigate the following types of objects that appear in mammogram images: calcifications, masses, architectural distortions and asymmetries. We will use the film mammographies that were provided from the mini mammographic database.

We believe that “deep learning” can provide a promising approach to machine learning in patient datasets and can be a useful component of a diagnostic support platform. The goal of this research is to classify the medical images from the mammographic database, with the use of stacked denoising autoencoders, experimenting with different values of the parameters, and resulting to the ones that give the smallest classification error.

The paper is organized as follows: Chapter 1 introduces the theory of general machine learning and classification. In chapter 2, we discuss the problem definition. Chapter 3 introduces the concept of autoencoders and stacked denoising autoencoders. Experimental results of classification error values for breast cancer mammography images are presented in chapter 4. Finally, in chapter 5, conclusions are drawn and future work is suggested.

CHAPTER 1

1 GENERAL MACHINE LEARNING

1.1. Introduction to Machine Learning

Machine learning is the field of research devoted to the formal study of learning systems. This is a very interdisciplinary field which borrows and builds upon ideas from statistics, computer science, engineering, cognitive science, optimization theory and many other disciplines of science and mathematics [1].

Over the past two decades Machine Learning has become one of the most important parts of information technology and with that, a rather central, though usually hidden, part of our life. With the ever increasing amounts of data becoming available, it is believed that smart data analysis will become a necessary ingredient for technological progress [2].

1.1.1. Definitions of Machine Learning

Two definitions of Machine Learning are offered [3]. The first is an informal and older one, expressed by Arthur Samuel: “Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed”, using algorithms that iteratively learn from data. A more modern definition was provided by Tom Mitchell: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Machine learning algorithms can figure out how to perform important tasks by generalizing from examples. This is often useful and can save cost where manual programming cannot [4].

1.1.2. Machine Learning Applications

Because of new computing technologies, machine learning today is not like machine learning of the past. While many machine learning algorithms have been used for a long time, the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development. Some examples of machine learning applications are [1]:

- Web page ranking: It is the process of submitting queries to a search engine and finding webpages relevant to the query, in order of relevance. Such information is provided by the structure and content of websites, the frequency of visits at a webpage, the frequency with which the users will follow the suggested links. It is becoming more and more essential to automate, with the help of machine learning, the process of designing a good search engine.
- Automatic translation: Using examples of translated documents, the machine can learn how to translate.
- Collaborative filtering: Internet bookstores or online video stores use information like past purchase, preferences or viewing decision of the users to predict future viewings and purchase habits and to suggest additional purchases. The key here are the decisions made by similar users, hence the collaborative nature of the process.
- Spam filtering: Whether an e-mail contains relative information or not. This is a problem of classification, which we are going to further analyze later in this assignment.
- Recognition applications like:
  - Face recognition: Given photos of people, the machine classifies their faces into known or unknown categories. It is important for the system to learn which features are relevant for the identification of a person.
  - Speech recognition: Given audio sequences, the system learns to annotate them with text.

- Handwriting recognition and named entity recognition (the problem of identifying entities, for example titles, places, names in documents).

- Other applications include cleaning robots, avatar behavior in video games, computer trackpads, direct marketing, fraud detection, new pricing models, real time ads, etc.

1.1.3. Machine learning algorithm

In Figure 1-1 we have examples of handwritten digits taken from telephone numbers.

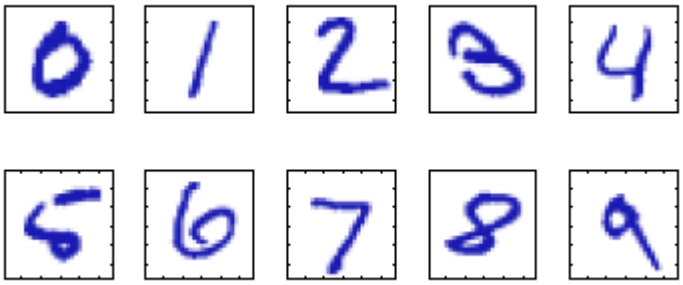


Figure 1-1: Examples of handwritten digits taken from telephone numbers.

As described in [5], each digit corresponds to a 28x28 image and can be represented by a vector x of 784 real numbers. The aim is to build a system that will take vector x as input and will produce the identity of digits 0…9 as output. The best results can be obtained through machine learning, adopting an approach, in which a large set of N digits (X_1, X_2, X_3, …, X_N) is used to train the parameters of an adaptive model. This set of digits is called a training set. The categories of the digits are known a priori, by labeling them manually and are represented with a target vector t.

The result of this machine learning algorithm is a function y(x), which takes vector x as input and gives vector y as output, encoded in the same way as target vector t. The exact form of the function y(x) is defined through the training phase (learning phase), based on the training data. Once the algorithm is trained, it can then be used to determine the identity of new digit images, called the test set. During the test phase, the machine learning algorithm categorizes the new images, based on the knowledge obtained by training phase. The ability to correctly categorize the new images is called generalization. This process of searching patterns in data is called pattern recognition and has a long and successful history [5].

1.1.4. Problems in machine learning

The main goal in pattern recognition is generalization beyond the training examples, because it is very unlikely to come across the same examples during test phase. A mistake commonly made is testing on the training data and thus having the illusion of success. The best strategy to solve this problem is to divide the input set to training and testing examples.

Another common problem in machine learning is overfitting. Intuitively, overfitting occurs when a machine learning algorithm fits the data too well. Overfitting can be understood if we decompose generalization error into bias and variance. Specifically, it is characterized by low bias and high variance. This problem can be prevented by using cross-validation on multiple models to test their accuracy [6].

Finally, an equally big problem in machine learning is the curse of dimensionality, i.e. various problems that arise while analyzing and organizing data at higher than three dimensions [7]. Specifically, [4], generalization becomes exponentially harder as the dimensionality increases. Usually, a big increase in the number of the data is required, in order to yield equally

good results [8]. The solution to the curse of dimensionality can be given with feature selection or dimensionality reduction.

1.1.5. Supervised & Unsupervised learning

The applications where the training data consists of examples of the input vectors, along with their corresponding target vectors, are known as supervised learning problems. Cases such as the handwritten digits' recognition problem, where the goal is to assign each input vector to one of a finite number of discrete categories, are called classification problems. We will talk about classification more thoroughly in the next section.

Other examples of classification tasks would be:

- A credit card company receives hundreds of thousands of applications for new cards. The applications contain information regarding several different attributes, for instance annual salary, debts, age, marital status etc. Categorize applications into those who have good credit, bad credit or fall into a grey area (thus require human analysis).
- Astronomers have been cataloguing distant objects in the sky. Label objects as stars, nebulas, galaxies, etc.
- An emergency room in a hospital measures a number of variables (e.g. blood pressure, age, cholesterol levels etc.) of newly admitted patients. Predict high-risk patients and discriminate them from low-risk patients.

In other pattern recognition problems, the training data comprises a set of input vectors x, without any corresponding target values. These problems are called unsupervised learning problems. The goals in these problems are:

→ Clustering: find patterns within the data

→ Density estimation: determine distribution of data within the input space

→ Visualization: Project data from a high-dimensional space to two dimensions

With unsupervised learning, there is no feedback based on the prediction results. Given that, it may seem mysterious to imagine what the machine could possibly learn [pattern].

Another machine learning method is semi-supervised learning, which uses both labeled and unlabeled examples (usually more unlabeled data, since it is cheaper and easier to obtain). Semi-supervised learning can be useful when the cost of labeling is too high and can be used with classification, regression and prediction. An example of semi-supervised training is the identification of a person's face on a web camera.

Finally, reinforcement learning is usually used on robotics, navigation applications and games. The algorithm of reinforcement learning discovers through trial and error the actions that yield the best results. It consists of three components: the agent (the learner), the environment (everything that the learner interacts with) and the actions (the things that the learner does). The goal for the algorithm is to choose the actions that will yield the best result, so its job is to discover the best policy for accomplishing this goal [9].

1.2. Classification

In everyday life, the word classification means categorizing and grouping things which share common characteristics. In machine learning, these groups are called classes. Classes can be defined as follows [10]:

- Classes can correspond to labels for different populations, e.g. populations of cats, dogs or other animals.
- Classes are outcome of prediction problems. In fact, classes must be predicted from known attributes or characteristics. For example, a common question on the prediction of rates is whether they will rise (class label = 1) or not (class label = 0).
- Classes are predefined by the attributes of the input space. For example, a manufactured item can be classed as faulty if some of its attributes are outside of the limits that have been determined.

The term classification is highly connected with a particular type of learning, during which examples of one or more classes, labeled with the name of the classes, are given to an algorithm as input. The algorithm produces a classifier, which assigns the characteristics of each example to each class label. When a new, unlabeled example is given to the algorithm, it tries to classify it, based on the example's properties [8].

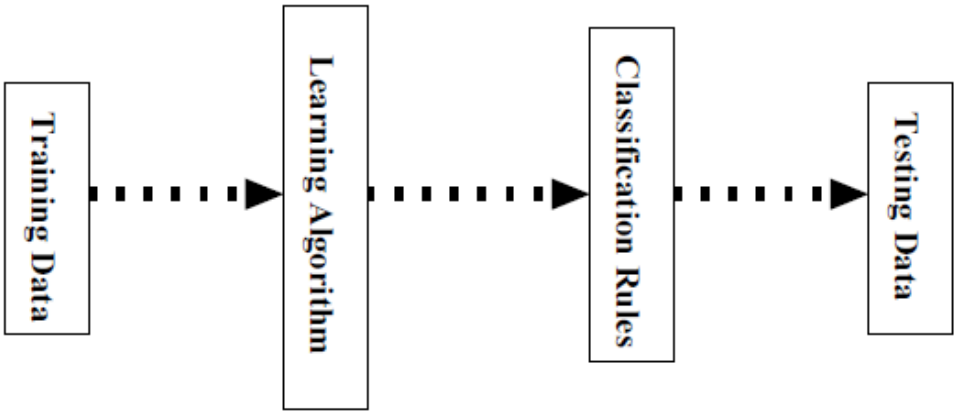


Figure 1-2: Classification process

A classifier is a system that takes as input a vector of feature values and gives as output a single value, the class. The criteria to choose a classifier are the following [4]:

- Representation: The classifier must be represented in a language understandable from the computer.
- Evaluation: An evaluation function is needed in order to distinguish the good from the bad classifiers.
- Optimization: We need a technique to search among the classifiers for the one that yields the best results.

The result of the classification is the division of the input space into regions that belong to a single class. The input space is defined by all possible combinations of all possible values of the input.

In Figure 1-3 [8] we can see two classes, - and +. The attributes of the example are real numbers and their range is $\pm\infty$. There are two different ways to divide our space: the first is with the bold lines, where the examples with $y<1$ and $x<1.5$ will be classified as +, while the second is with the dashed line, where the examples below the line will be classified as + and those above it as -.

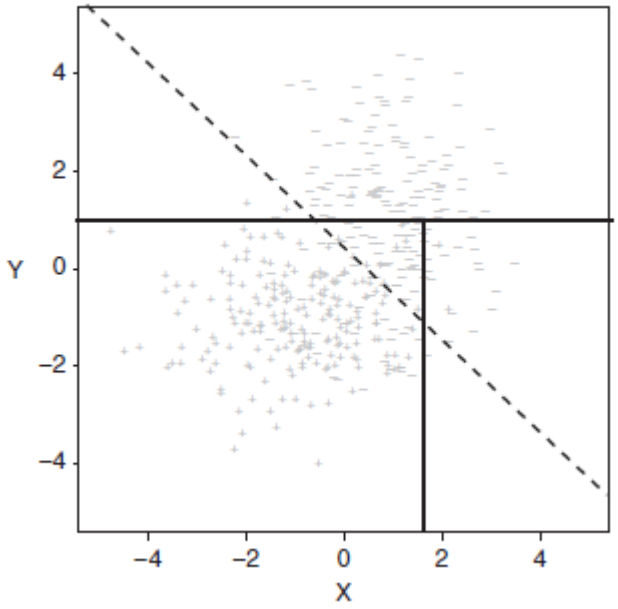


Figure 1-3: Classification. Division of the input space.

CHAPTER 2

2 PROBLEM DEFINITION

Breast cancer is a disease where abnormal cells grow in an uncontrolled fashion [11]. Breast cancer is the most common form of cancer among women and is the second leading cause of death after lung cancer. The American Cancer Society [12] estimated that, in 2013, approximately 232,340 women in the US were diagnosed with breast cancer and about 39,320 women [13] died from breast cancer (Figure 2–1). The World Health Organization's International Agency for Research on Cancer in Lyon, France, has estimated that more than million women worldwide died to breast cancer each year [11].

Table 1. Estimated New Female Breast Cancer Cases and Deaths by Age, US, 2013*

Age (Yrs)	In Situ Cases	Invasive Cases	Deaths
<40	1,900	10,980	1,020
<50	15,650	48,910	4,780
50-64	26,770	84,210	11,970
65+	22,220	99,220	22,870
All ages	**64,640**	**232,340**	**39,620**

*Rounded to the nearest 10.

Source: Total estimated cases are based on 1995-2009 incidence rates from 49 states as reported by the North American Association for Central Cancer Registries. Total estimated deaths are based on data from US Mortality Data, 1995-2009, National Center for Health Statistics, Centers for Disease Control and Prevention.

American Cancer Society, Surveillance and Health Services Research, 2013

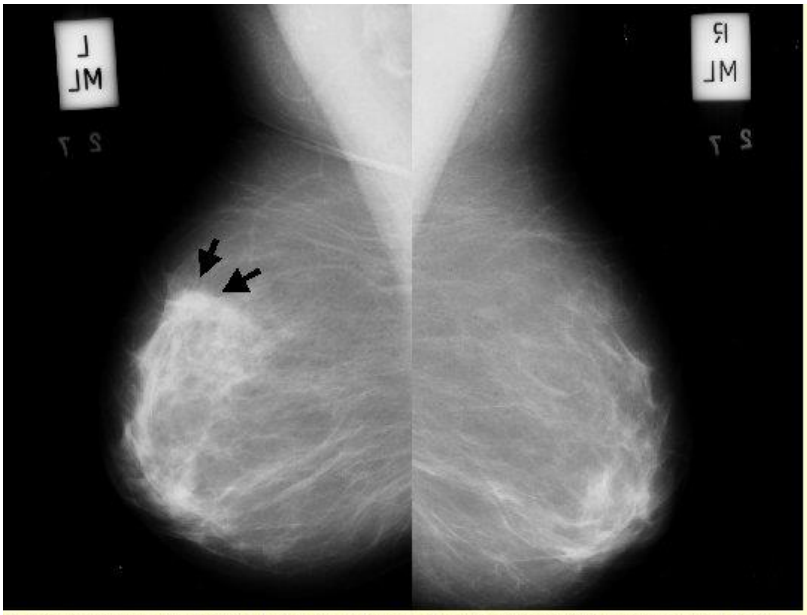
Figure 2-1: Estimated female breast cancer

Early detection reduces breast cancer mortality by approximately 25%. The early detection is possible thanks to screening mammography, the main goal of which is to capture cancerous lesions before any clinical symptoms appear (when mammograms are taken from patients where there is not suspicion of illness) [14]. According to [11], mammography is a specific type of imaging that uses a low-dose X-ray system, high contrast and high resolution film for the examination of the breasts. Usually, a mammographic abnormality is followed up by additional imaging studies, such as ultrasound, and if the lesion still appears suspicious, then it is sent for biopsy.

Breast cancer is usually associated with [15]:

- Asymmetries between images of left and right breasts
- Distortion of the normal architecture of the breast tissue
- Presence of masses in the breast, which can be classified as benign or malignant
- Presence of microcalcifications in the breast

As Steven Halls mentions in [16] an asymmetrical breast tissue is an observation made with respect to the same area on the other breast. Asymmetrical breast density should be considered suspicious when it is also associated with a clinically palpable breast asymmetry. Otherwise, a certain amount of asymmetrical breast tissue should be considered a normal variation which occurs in some women. An asymmetric density refers to an obscured view in part of the breast, which is visible from only one projection or view of the mammogram (Figure 2-2).



The asymmetric area of the left breast has its density concentrated centrally and has the appearance of mass formation. Here, it was found to be associated with cancer.[2]

Figure 2-2: Asymmetric density associated with cancer

An architectural distortion on a mammogram is a disruption of the normal pattern structures that are normally seen on a breast (Figure 2-3).

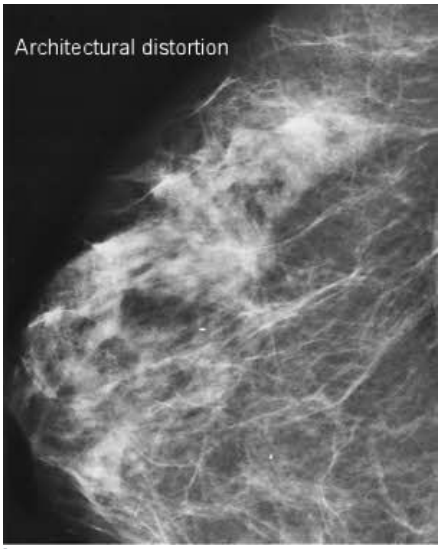


Figure 2-3: Asymmetric distortion

Masses are space-occupying lesions [15] that are seen on at least two projections. They are described according to three characteristics: their shape or contour, margin and density (fatty, low, iso-dense, high) (Figure 2-4).

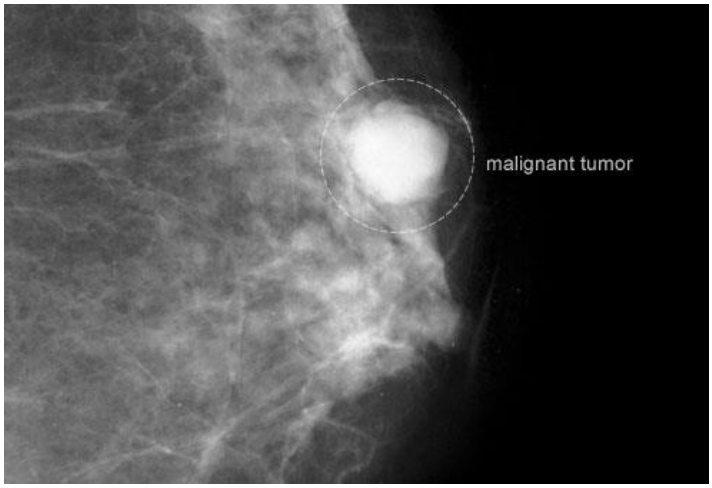


Figure 2-4: Malignant tumor

Finally, microcalcifications represent calcium deposits located in the breast tissue and highly indicate breast cancer. They appear as small, bright objects that stand out from the surrounding tissue with diameter from 0.1 to 3 mm (Figure 2-5) [15], [14].

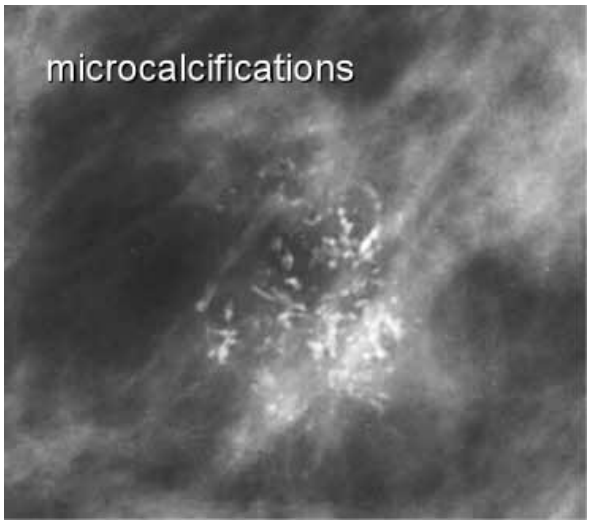


Figure2-5: Breast microcalcifications

However, the visual interpretation of a mammography is a tedious and fatiguing process that requires usually a magnifying glass [12]. When radiologists need to diagnose a mass in a mammogram, they look for some significant features that discriminate malignant from benign masses. These visual features -which are based on shape, size and margin - could have different interpretation based on radiologists' opinion and experience. By applying computer process imaging techniques on digitalized mammograms and then extracting features from suspicious regions, we can help the medical discover more easily a tumor and improve their facilities.

According to [12], two systems have been developed to help radiologists read mammograms:

- CADe (Computer Aided Detection), which has improved radiologists' accuracy in discovering tumors.

- CADx (Computer Aided Diagnosis), which classified the suspicious regions into benign or malignant, in order to help the radiologists to continue with the next step.

Most diagnosis algorithms of CADx consist of one stage with five steps: preprocessing, segmentation, feature extraction, feature selection, and classification.

In this experimental study, we deal with the problem of feature extraction with stacked denoising autoencoders (which will be analyzed in the next chapter) in digital mammograms, because the feature extraction process is an important key for early detection of breast cancer [11]. We will use the digital mammograms that were provided from online mammogram database (MIAS Database) [17]. We will investigate the following types of objects found in mammograms:

- Calcifications
- Well defined – circumscribed masses
- Other, ill-defined masses
- Architectural Distortions and
- Asymmetries

We will experiment on a number of learning parameters, train the model for each combination of settings and, thus, discover the best parameters that will lead, on the next step, to the correct classification of these medical images. We will use a neural network as a classifier and we will draw conclusions based on the classification errors.

CHAPTER 3

3 AUTOENCODERS

3.1. Simple autoencoders

3.1.1. The structure of a simple autoencoder

An autoencoder, or auto-associator or Diabolo Network, is an artificial neural network, used for learning efficient codings (Figure 3-1) [18]. It has an input layer, which represents the original data or input feature vectors (e.g. pixels in images), one or more hidden layers that represent the transformed features and an output layer, which is the reconstruction of the input. So an autoencoder attempts to reproduce its input x, by encoding it to a hidden layer h, and decoding it to a representation z. Through that representation, it attempts to reconstruct the input signal.

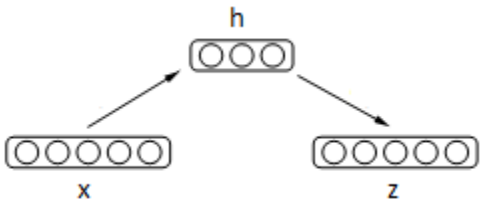


Figure 3-1: Autoencoder

$h(x) = sigm(W_x + b)$ (hidden representation h), parameterized by $\theta = \{W, b\}$

$z(x) = sigm(W'h(x) + b')$ (representation z of input x), with $\theta' = \{W', b'\}$

where sigm is a non-linearity function, the sigmoid, W is a d' × d weight matrix and b is a bias vector.

As decribed in [19], an autoencoder takes the input x ∈ [0, 1]^d, encodes it to a hidden representation h ∈ [0, 1]$^{d'}$ and decodes it to a reconstruction z ∈ [0, 1]^d of the input x, so z should be seen like the prediction of x, given the code h. In general, z should not be seen as an exact reconstruction of x, but more as the parameters of a distribution p(X|Z=z) that may generate x with a high probability [20]. The goal is to optimize the parameters (W1, W2, b1, b2), so as to minimize the average reconstruction error.

$$\theta^\star, \theta'^\star = \arg\min_{\theta,\theta'} \frac{1}{n}\sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}\right)$$

$$= \arg\min_{\theta,\theta'} \frac{1}{n}\sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, g_{\theta'}(f_\theta(\mathbf{x}^{(i)}))\right)$$

where L is a loss function.

The reconstruction error can be measured in many ways, depending on the distribution of the input data. The traditional mean squared error L(xz)=||x-z||^2 could be used. If the input is a bit vector or a vector of bit probabilities (Bernoullis), then we can use cross-entropy of the reconstruction.

$$L_H(\mathbf{x}, \mathbf{z}) = -\sum_{k=1}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

«Breast cancer classification of mammographies using stacked denoising autoencoders».

Since h is considered to be a compression of x with loss, it will be a lossy compression of every x. This is a characteristic of the generalization of an autoencoder: when the test examples are from the same distribution as the training examples we have low reconstruction error, but when our samples are taken randomly from the input space, the reconstruction error is high [19].

Many autoencoders structures exist: with or without tied weights*, with deterministic stochastic variables, with various activation functions etc. [21]. In this chapter, we are going to describe how an autoencoder works and we will continue with denoising and stacked denoising autoencoders.

3.1.2. Training process of an autoencoder

Training autoencoders is said to be easier than training RBMs [21], hence they have been used as building blocks for training deep networks, with the representation of the k_{th} layer being the input of the $k+1_{th}$ layer and having been trained before it. After some layers have been trained, the parameters are used as initialization for a network optimized with a supervised training criterion. An autoencoder is trained, usually applying the stochastic gradient descent method, which, while effective, it creates basic problems when the network has many hidden layers. In contrast, the greedy layer-wise training has been shown to have significantly better results and better generalization on many tasks.

Let's suppose we have some unlabeled training examples: {x(1), x(2),x(3),…}, where $x(i) \in Rn$. As described in [22], an autoencoder is basically an unsupervised learning algorithm, that is setting the target values to be equal to the input values, applying backpropagation, i.e. it uses y(i) = x(i).

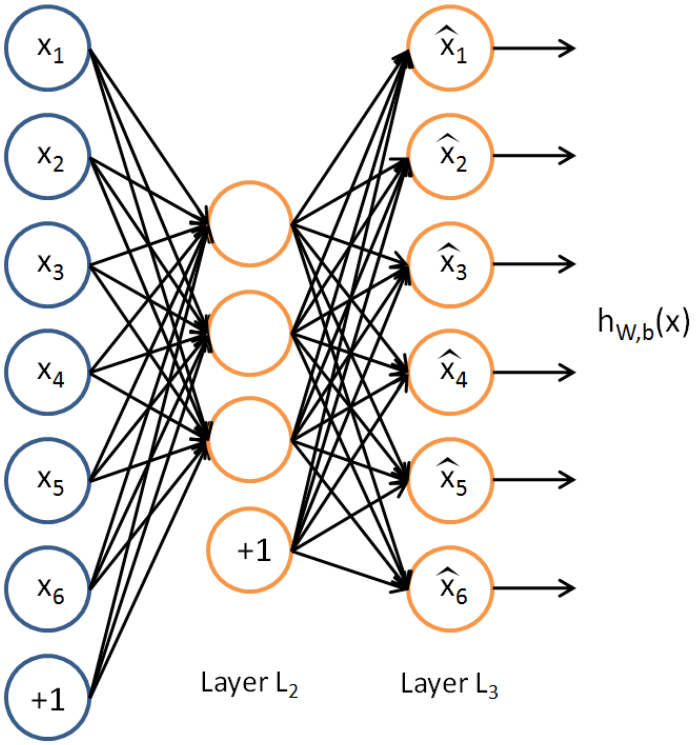


Figure 3-2: An autoencoder

* When the weight of the representation z (W2) is the matrix transpose of the weight of the hidden representation h (W1), i.e. $W2 = W1^T$, this is referred to as tied weights [19, 23].

The autoencoder tries to learn an approximation to the identity function, so that the output z is similar to the input x. In order to prevent the autoencoder from simply replicating the input and create a representation that exploits the statistical regularities in the training set, we have to place constraints on the network, besides minimizing the reconstruction error. This way we can have some interesting results concerning our data and useful representations. One of the constraints that can be placed is changing the number of the hidden layers/units.

For example, let's suppose that our inputs x are the pixel intensity values taken by a 10x10 image (i.e. 100 pixels), so n=100, and, also that there are s2=50 hidden units. Given that, the autoencoder needs to learn a "compressed" representation of its 100-pixel input. Furthermore, if the input data is structured and there are correlations between the features, then the autoencoder is able to find out these correlations [22].

The code h should be a distributed representation that can capture the main factors of variation in the data, similar to the way principal component analysis works. Principal component analysis is "a statistical procedure that converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components". The number of the principal components equals to the initial number of the variables. The transformation happens in such a way, so that the first principal component has the largest possible variance, meaning that it contains the most of the variability of the data, and each following component has the highest variance, providing that it is orthogonal to the previous components [24].

So, in a similar way, if we have one linear hidden layer and we use the mean-squared error for training, the input is projected by the k hidden units on the first k principal components of the data. However, if the hidden layer is not linear, the behavior of the autoencoder is different from principal component analysis, since the input distribution is multi-modal. This is even more important when we move to denoising and stacked denoising autoencoders [19, 21].

However, even when the number of hidden layers is larger (in this case maybe more than 100 – called overcomplete), we can still discover correlations and structure in our data. Indeed, it has been proved [21] that non-linear autoencoders, trained with stochastic gradient descent, that have more hidden units than input give more useful representations.

We will take a moment to explain and define what a "useful" representation means. A good representation is one that is helpful for addressing tasks of interest, and will help the system achieve higher performance than it would before the formation of the representation. In other words, a useful representation yields a classifier with better results.

Other constraints that can be imposed on the network are the addition of sparsity (setting hidden units zero or close to zero), the activation and deactivation of the hidden layers and the addition of randomness during the transformation of input data to reconstruction [19] (encoding process), which is described below.

3.2. Denoising autoencoders

A denoising autoencoder works like a simple autoencoder. However, in order to prevent the autoencoder from learning the identity function, we train it to reconstruct the input from a corrupted version of it. The corruption process is stochastically applied to the input of the autoencoder. So the task of reconstruction of corrupted input to uncorrupted is called denoising [23].

Accordingly, we can change the definition of a useful representation into the following [20]: "a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input".

As mentioned in [19, 21], the denoising autoencoder does two things: it encodes the input (preserving the information concerning it) and it tries to reconstruct it, undoing the corruption process that has been applied. This can be done by analyzing the statistical dependencies between the inputs.

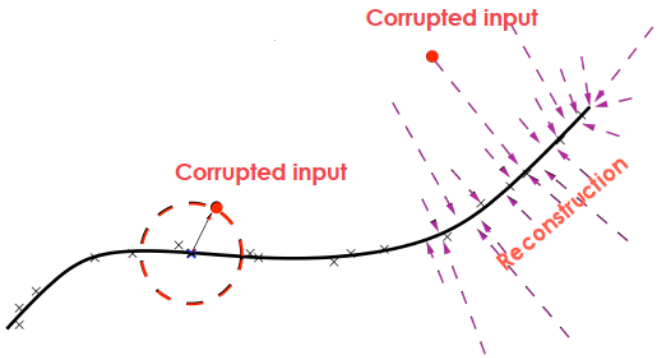


Figure 3-3: Reconstruction process

With denoising autoencoders we investigate an additional criterion [23]: the robustness to partial corruption of the input, i.e. insensitivity to small random perturbations. Corrupted inputs yield almost as useful representations. The idea behind this criterion is the following: a useful representation is known to capture stable structures in the form of statistical regularities and dependencies in order to recognize a possible distribution of the input. For high dimensional input, such as images in our case, these structures are expected to be captured by evidence that is collected from many input dimensions. So, they should be recovered from partial observation only. That derives from the fact that humans are able to recognize images which are partially hidden or corrupted.

We will train the autoencoder to reconstruct a clean input from a partially corrupted one. This can be done by first corrupting the input to get a partially destroyed version. The corruption process goes as follows: a fixed number of components of the input is chosen randomly and their value is set to zero, while the others are not touched. Thus, all the information concerning these components is erased, so the autoencoder has to “fill in” these newly introduced “blanks”. After that, the corrupted input x is mapped to a hidden representation h and is reconstructed to output z. Like before, the parameters are optimized in order to minimize the average reconstruction error, i.e. to have z be as close as possible to the undestroyed input x.

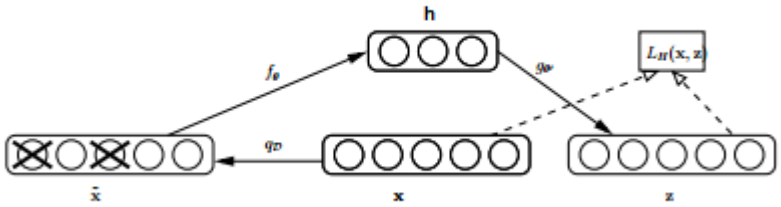


Figure 3-4: An example x is corrupted to x~. The autoencoder then maps it to h and attempts to reconstruct x.

First we corrupt the input x into $\tilde{x}$ by means of a stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$. Then we map the corrupted input $\tilde{x}$ to a hidden representation $h = s(W\tilde{x} + b)$, and we decode it to the reconstruction z (Figure 3-4). Our goal is, as with simple autoencoders, to minimize the reconstruction error, i.e. to have a reconstruction z that will be as close to x as possible. The

basic difference now is that z is a deterministic function of $\tilde{x}$ instead of x. Like before, the reconstruction error is the cross entropy loss:

$$L_H(x,z) = H((B,x) \parallel (B,z))$$

Or the squared error loss:

$$L_2(x,z) = \parallel x - z \parallel^2$$

The parameters are initialized randomly and are optimized with stochastic gradient descent [20].

Using the stochastic gradient descent algorithm, we will pick an input sample from the training set, we will corrupt it and then try to reconstruct it, by taking a gradient step. In this way, we can prevent the autoencoder from learning the identity function, without imposing additional criteria.

In the greedy layer-wise training, the procedure for the denoising autoencoder is similar to the one used for the simple autoencoder, with the difference that each layer receives as input the uncorrupted output of the previous layer and that the corruption process is used only during training [23].

We will go through some simple corruption processes that are useful for learning higher representations by stacking denoising autoencoders.

1. Additive isotropic Gaussian noise (GS): This is a very common corruption model for real valued inputs. $GS = \tilde{x}|x \sim N(x, \sigma^2 I)$
2. Masking noise (MN): This is a common corrupting technique, through which some components (randomly chosen) that are considered missing are forced to zero, or their value is replaced by a default value.
3. Salt and pepper noise (SP): With this noise model, a fraction of the elements (again randomly chosen) is set to their minimum or maximum possible value (0 or 1) depending on the result of a coin flip. The salt and pepper noise method is better for binary input, such as black and white images.

In both the masking noise and salt and pepper noise methods, some elements are corrupted, while the rest are left untouched. Thus, the denoising process becomes possible thanks to dependencies between input dimensions, that must be captured.

3.3. Deep autoencoders

A deep autoencoder is a special type of a deep neural network. An autoencoder is considered to be deep when its number of hidden layers is greater than one. The autoencoders can also be used as building blocks for building and initializing a multi-layered deep network, which is called a stacked autoencoder, where autoencoders are stacked on top of each other and are trained in a greedy layer-wise fashion [prediction]. According to the deep stacking design, simple modules or functions – in our case autoencoders – are composed and then stacked on top of each other, in order to learn complex representations [25].

In such deep architectures, we are facing fundamental problems regarding their training methods, though they are often effective. In back-propagation, the errors are back-propagated to the first layers, hence they become minuscule and thus the training is not as effective. The result is that the network reconstructs almost always the average of the training data. This leads to very slow learning and poor solutions. The solution to this problem is using initial weights that approximate the final solution. The process of finding these weights is called pretraining [26]. The pretraining method in stacked autoencoders consists of training each layer unsupervised on the activations of the layer below one after another [27].

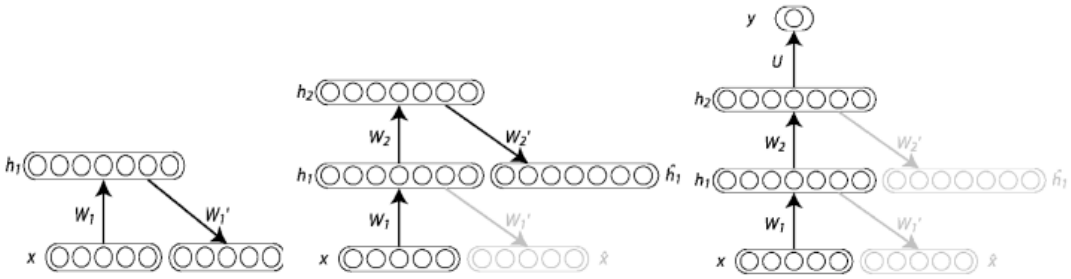


Figure 3-5: Stacked autoencoder

As we can see in figure 3-5, a stacked autoencoder consists of a number of layered modules, each one of which includes a single hidden layer and two trainable sets of weights. “The first module of the stacked autoencoder consists of a linear layer with a set of linear input units, a hidden nonlinear layer with a set of nonlinear units and a second linear layer with a set of linear output units” [25]. In our case, the input unit correspond to pixels of the images. The linear output layer represents the targets of classification.

W, which is the weight matrix of the first modules, connects the linear input and the hidden nonlinear layer. U, which is the weight matrix of the last modules, connects the hidden nonlinear and the linear output layers [25].

The training process of a stacked autoencoder is similar to the one used in deep belief networks [21]:

1. The first layer is trained with an unsupervised criterion as an autoencoder and the reconstruction error of the input is minimized.
2. The outputs of the hidden units are used as inputs for the next layer, which is also trained with an unsupervised criterion as an autoencoder.
3. Step 2 is repeated for all hidden layers.
4. The output of the last hidden layer is used as input to a supervised layer and the layer’s parameters are initialized.
5. The parameters and the global reconstruction error are fine-tuned with the supervised criterion.

3.4. Stacked denoising autoencoders

In this project, we use stacked denoising autoencoders in order to initialize a deep network that will yield a useful representation.

Let’s clarify that the corruption of the input is applied only on the initial denoising-training of each layer, so that it may extract some useful features. Once the mapping f_θ is learnt, it will then be used on uncorrupted input. In fact, the representation that will be used as a clean input for the training of the next layer is uncorrupted.

In Figure 3-6 we can see the complete process of training and stacking denoising autoencoders. After training a denoising autoencoder (first module), its encoding function f_θ is used on clean input (second module). With the yielded representation, a second level autoencoder can be trained to learn a second level encoding function $f_\theta^{(2)}$. Then, the procedure is repeated (third module) [20].

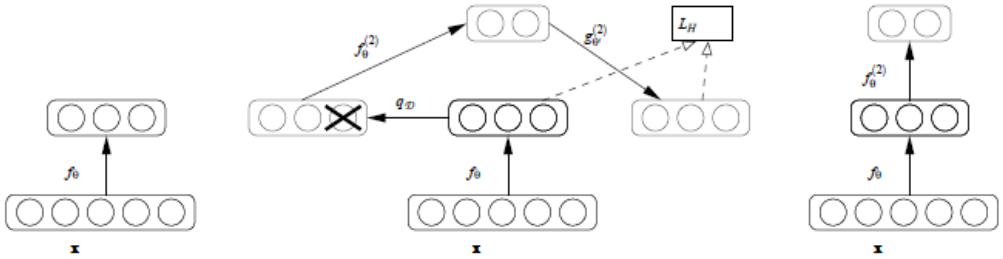


Figure 3-6: Stacking denoising autoencoder

As described in [28], the training process of the stacked denoising autoencoders has two steps: unsupervised pre-training and supervised fine-tuning. In the first step, which is described more thoroughly on the next section, we train each layer as a denoising autoencoder with a visible input and a hidden representation. With the first k layers trained, the output can be used as input for the k+1 layer above. The second step of the training process can start after all the layers have been trained. This is a back-propagation step, which uses supervised learning algorithms, in order to adjust the parameters of the model.

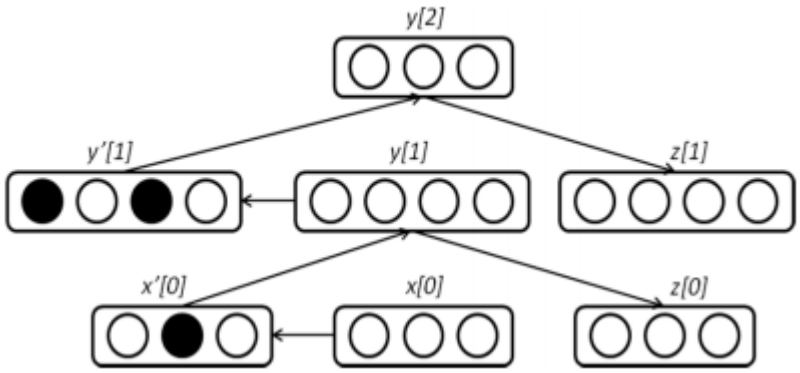


Figure 3-7: Example of stacked denoising autoencoder. The input x is mapped to y and then reconstructed to z. Output y is used as input to the next layer and the same steps are repeated until a deep network is formed.

3.4. Greedy layer-wise training

The greedy layer-wise training algorithm is a generative model with many layers of hidden causal variables [29]. The idea of greedy layer-wise training is to learn a hierarchy of features, one level at a time, with an unsupervised criterion. With each iteration of unsupervised feature learning, one layer of weights is added to the deep network. The learned representation at each layer is composed by the representations learned on the previous layer. This layerwise process can also be applied with a supervised criterion and it is called the greedy layerwise supervised pre-training [30].

The reason for which the greedy layerwise training is successful is that with the unsupervised pretraining it is easier to optimize the deep network, i.e. to better initialize the weights of the layers.

There are three stages in the process of greedy layer-wise training [29]:

1. Pre-training one layer at a time in a greedy way
2. Using unsupervised learning at each layer to preserve information regarding the input
3. Fine-tuning the network with respect to the ultimate criterion of interest

When using the greedy layerwise algorithm for training autoencoders, the training procedure is the following [21]:

1. The first layer is trained with an unsupervised criterion as an autoencoder, so as to minimize the reconstruction error of the input.
2. The outputs are used as input for the next layer, that is also trained as an autoencoder.
3. Steps 1 and 2 are repeated for the initialization of the desired number of layers.
4. The output of the last hidden layer is used as input to a supervised layer and its parameters are initialized.
5. All the parameters of the deep network are fine-tuned with respect to the supervised criterion.

The following pseudocode algorithm is taken from [29] and shows the process of greedy layer-wise training, when each layer is trained as an autoencoder:

Algorithm 3 `PreTrainGreedyAutoEncodingDeepNet`$(\widehat{p}, C, \epsilon, L, n, W, b)$

Initialize all layers except the last in a multi-layer neural network, in a purely unsupervised way, with the greedy layer-wise procedure in which each added layer is trained as an auto-associator that tries to reconstruct its input.

$\widehat{p}$ is the training distribution for the network

$C = -\log P_\theta(u)$ is a reconstruction error criterion that takes θ and u as input, with θ the parameters of a predicted probability distribution and u an observed value.

ϵ is a learning rate for the stochastic gradient descent in reconstruction error

L is the number of layers to train

$n = (n^0, \ldots, n^L)$, with n^0 the inputs size and n^i the number of hidden units in each layer $i \geq 1$.

W^i is the weight matrix for level i, for i from 1 to L

b^i is the bias vector for level i, for i from 0 to L

- initialize $b^0 = 0$.
- define $\mu^0(x) = x$.

for $\ell = 1$ to L **do**

- initialize $b^\ell = 0$.
- initialize temporary parameter vector $c^\ell = 0$.
- initialize W^ℓ by sampling from uniform$(-a, a)$, with $a = 1/n^{\ell-1}$.
- define the ℓ-th hidden layer output $\mu^\ell(x) = \text{sigm}(b^\ell + W^\ell \mu^{\ell-1}(x))$.
- define the ℓ-th hidden layer reconstruction parameter function, e.g. in the binomial case $\theta^\ell = \text{sigm}(c^\ell + W^{\ell\prime} \mu^\ell(x))$ is the vector of probabilities for the each bit to take value 1.

while not stopping criterion **do**

for $i = 1$ to $\ell - 1$ **do**

- compute $\mu^i(x)$ from $\mu^{i-1}(x)$.

end for

- compute $\mu^\ell(x)$ from $\mu^{\ell-1}(x)$.
- compute reconstruction probability parameters θ^ℓ from $\mu^\ell(x)$.
- compute the error C in reconstructing $\mu^{\ell-1}$ from probability with parameters θ^ℓ.
- compute $\frac{\partial C}{\partial \omega}$, for $\omega = (W^\ell, b^\ell, c^\ell)$
- update layer parameters: $\omega \leftarrow \omega - \epsilon \frac{\partial C}{\partial \omega}$

end while

end for

Figure 3-8: Pseudo-code for a deep network obtained by training each layer as an auto-encoder

CHAPTER 4

4 EXPERIMENTATION

In this section, we first introduce the database used for our experiments. Then we present the results yielded from the training of the stacked denoising autoencoders and the classification of the images.

4.1. Data set

Our input set is the MIAS mini mammographic database [database]. It consists of 322 .pgm images, which are 1024x1024 pixels and have been reduced, by request, from the original MIAS database, to 200 micron pixel edge. The database provides the following details:

- In the first column there is the MIAS database reference number.
- In the second column, the character of the background tissue is given:
  F – Fatty
  G – Fatty – glandular
  D – Dense – glandular
- The third column gives us the class of abnormality which is present on the images:
  CALC – calcifications
  CIRC – well defined/circumscribed masses
  SPIC – speculated masses
  MISC – other, ill-defined masses
  ARCH – architectural distortion
  ASYM – asymmetry
  NORM – normal
- In the fourth column, we have the severity of the abnormality:
  B – Benign
  M – Malignant
- The fifth and sixth column gives the x and y coordinates of the abnormality in the image
- In the seventh column, we are given the approximate radius (in pixels) of a circle enclosing the abnormality

The database is arranged in pairs of images. The even numbers represent the left mammograms and the odd the right mammograms.

As mentioned above, the size of the images is 1024x1024 pixels. Due to computational requirements, all images have been resized to 614x614 pixels. Examples of MIAS mini mammographic database are shown in figure 5-1.

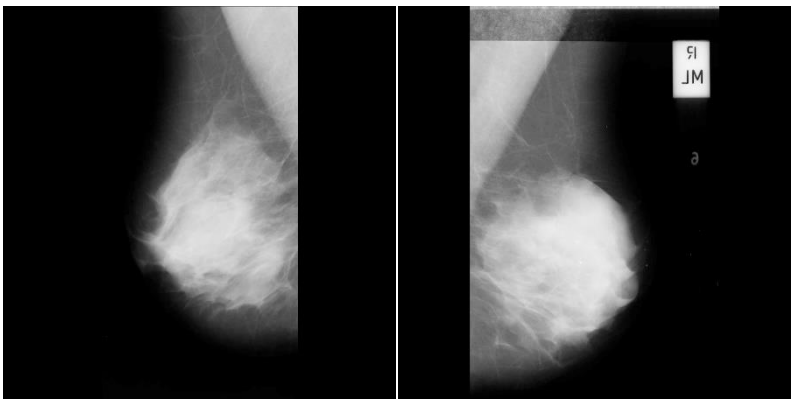


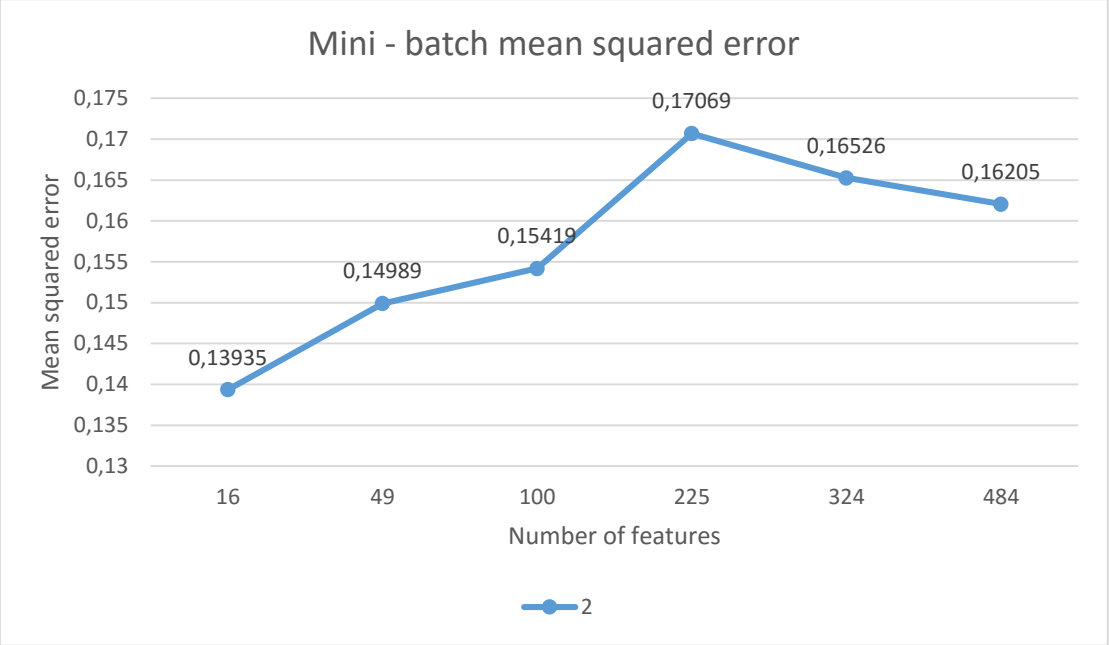
Figure 5-1: Examples of MIAS mini mammographic database

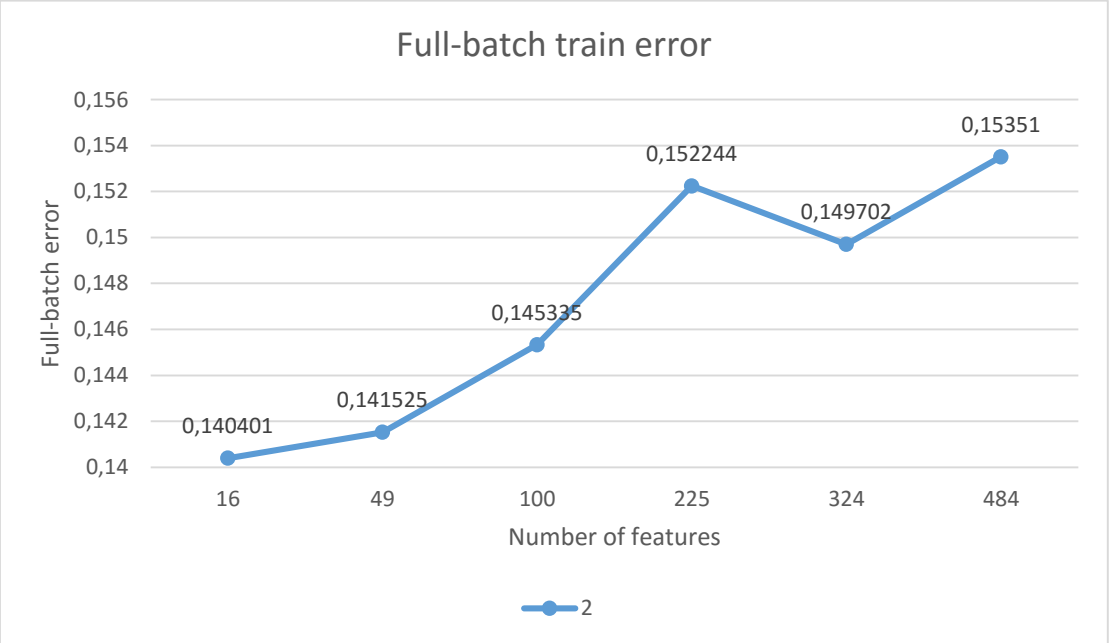
4.2. Experiments

While training the stacked denoising autoencoders, we experimented with different parameters, in order to be able to make comparisons and discover the ones that yield the best results. We trained the model with the combination of 1,3,5 epochs, 2,5,10,29,58,145,290 number of batchsize, and 16, 49, 100, 225, 324, 484 number of features. We will now present the results of each combination of parameters and compare the mini-batch mean squared error and the full-batch training error.

4.2.1. First set of experiments

The first set of experiments was done changing the number of features and keeping the number of epochs to 1 and the number of batchsize to 2.





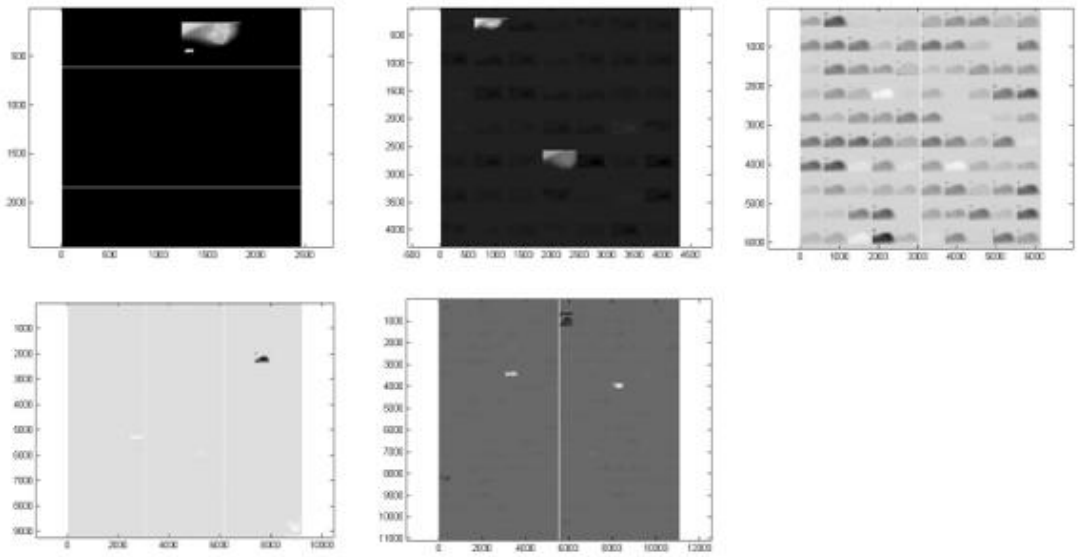
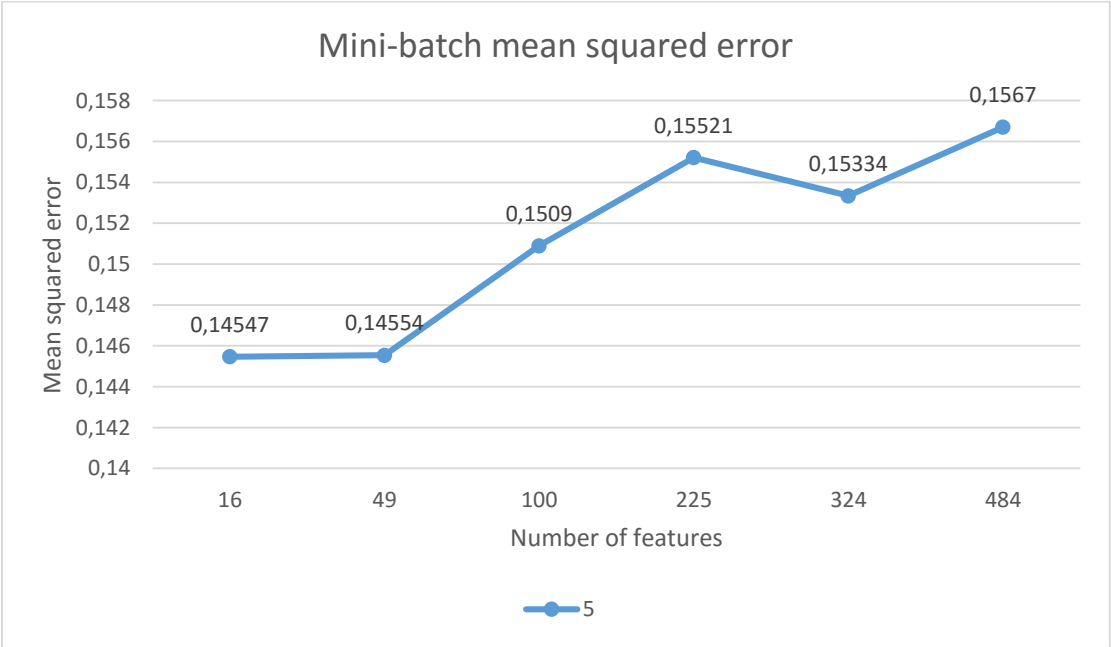
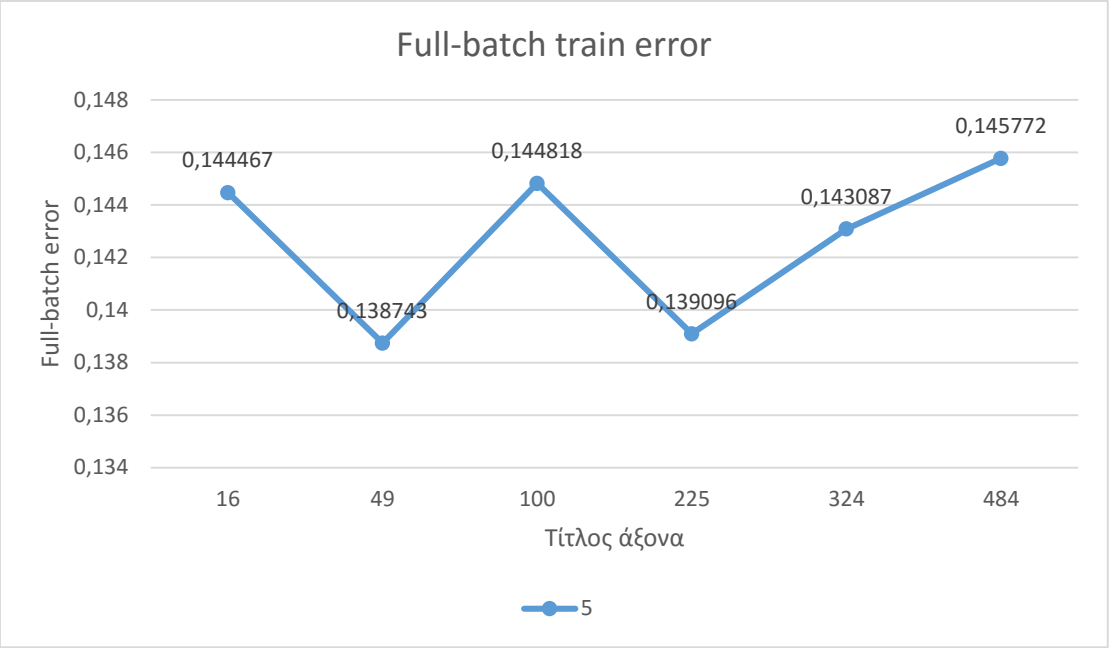


Figure 5-2: Features extracted: 1 epoch, 2 batchsize and 16, 49, 100, 225, 324 features (left to right).

We can see that as the number of features increases, both the mini-batch and full-batch training errors increase.

The next experiment was done changing the number of features and keeping the number of epochs to 1 and the number of batchsize to 5.





Here we can see that with the increase of the number of features, the mini-batch mean squared error increases too, whereas the full batch training error shows a different behavior, being smallερ with 49 and 225 features.

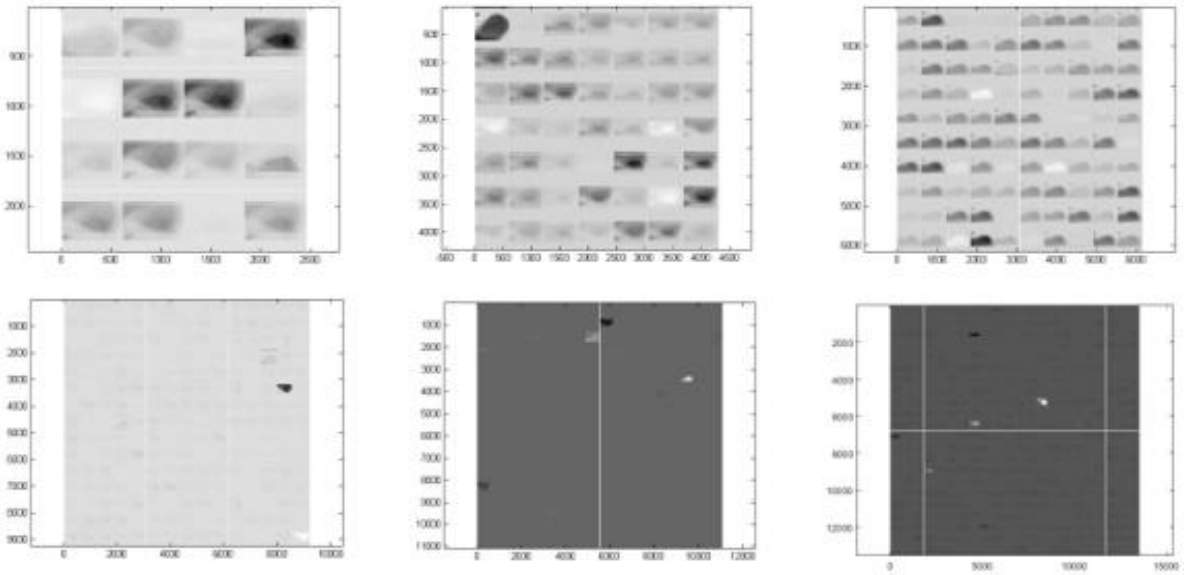
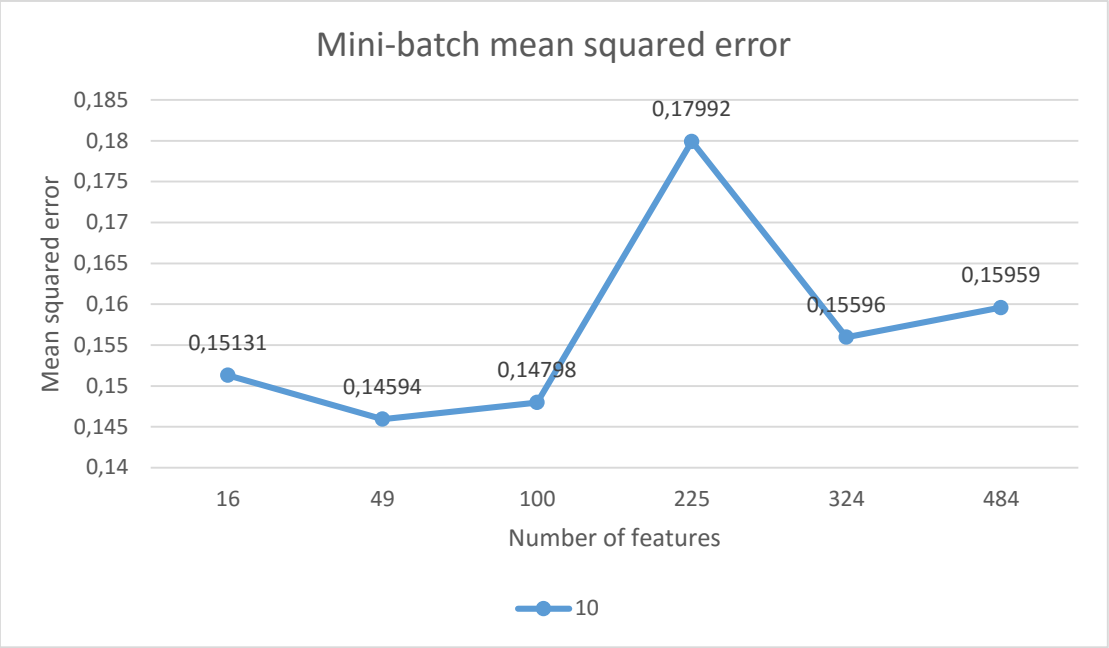
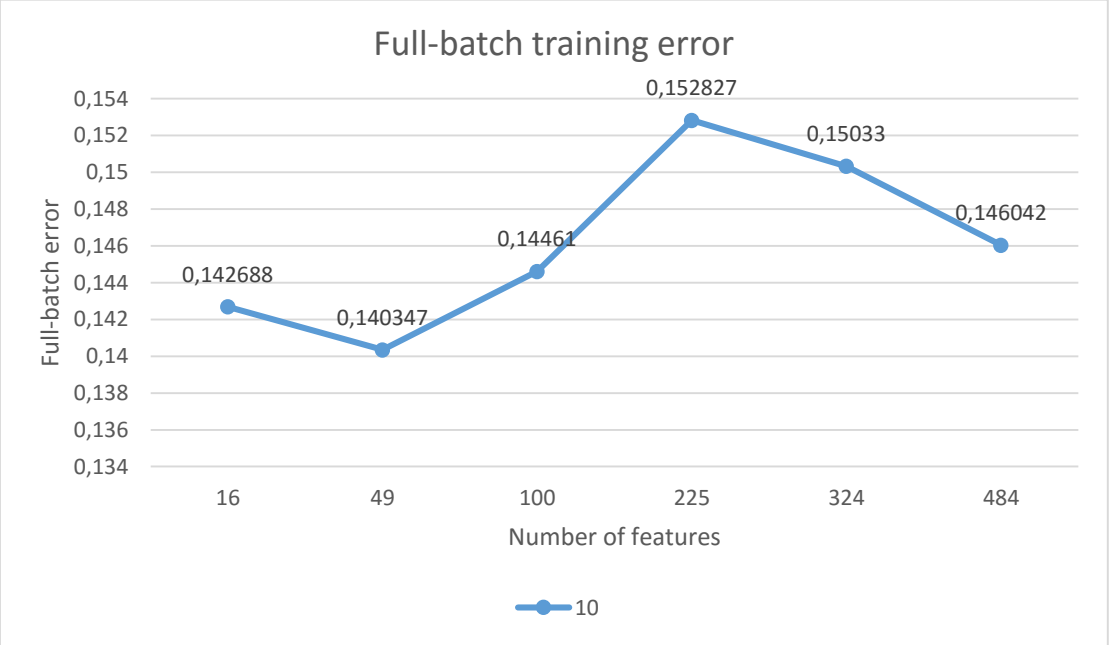


Figure 5-3: Features extracted: 1 epoch, 5 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

Next, our experiments were done using 10 number of batchsize.





There is a common pattern in both the diagrams. When the number of features is 49, we have the minimum error, while when the number of features is 225, the error is maximum.

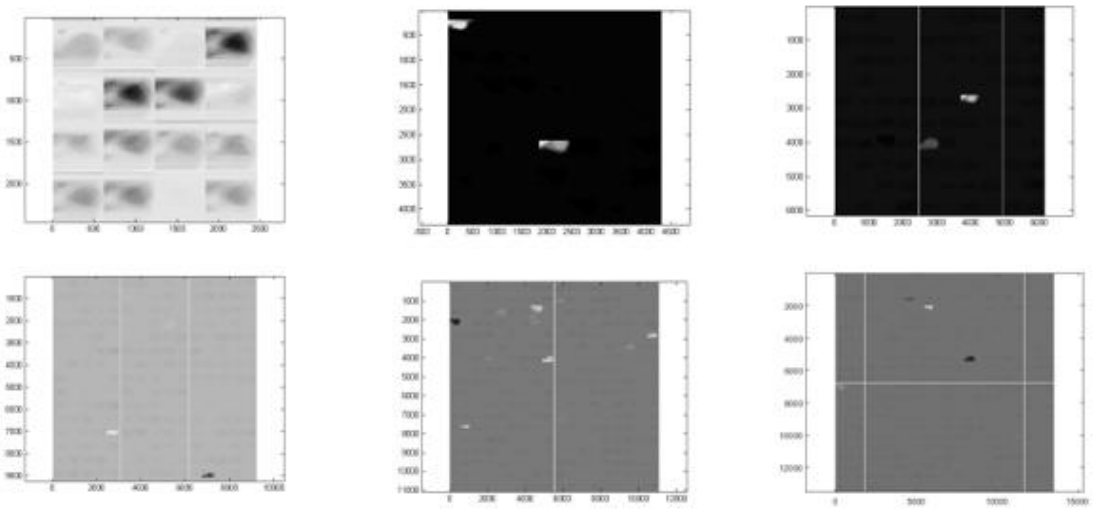
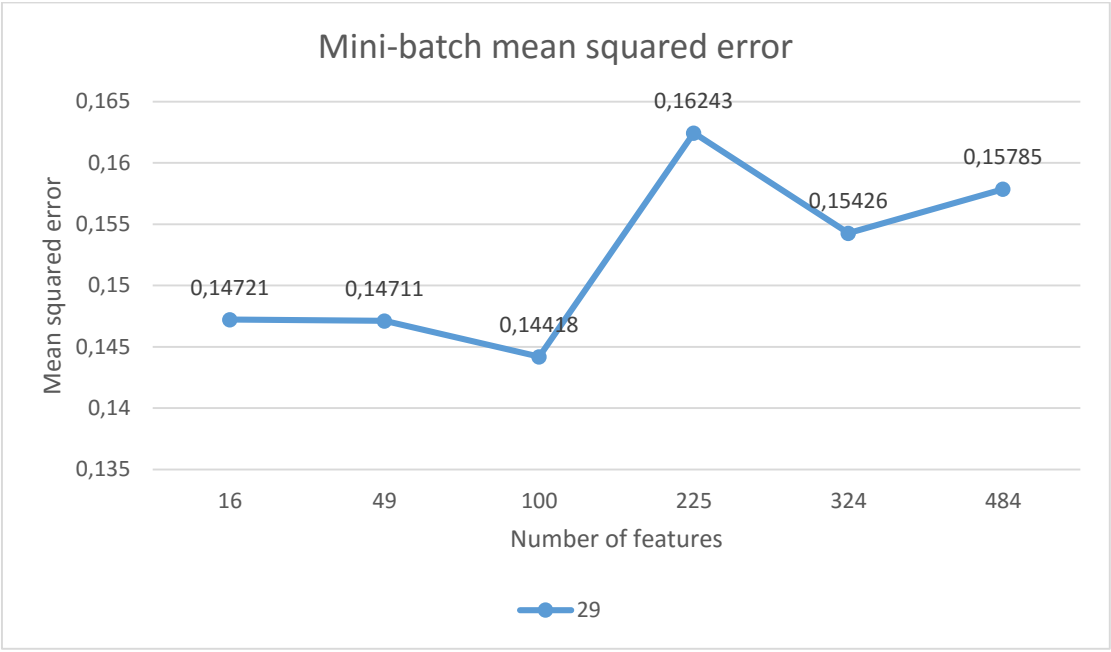
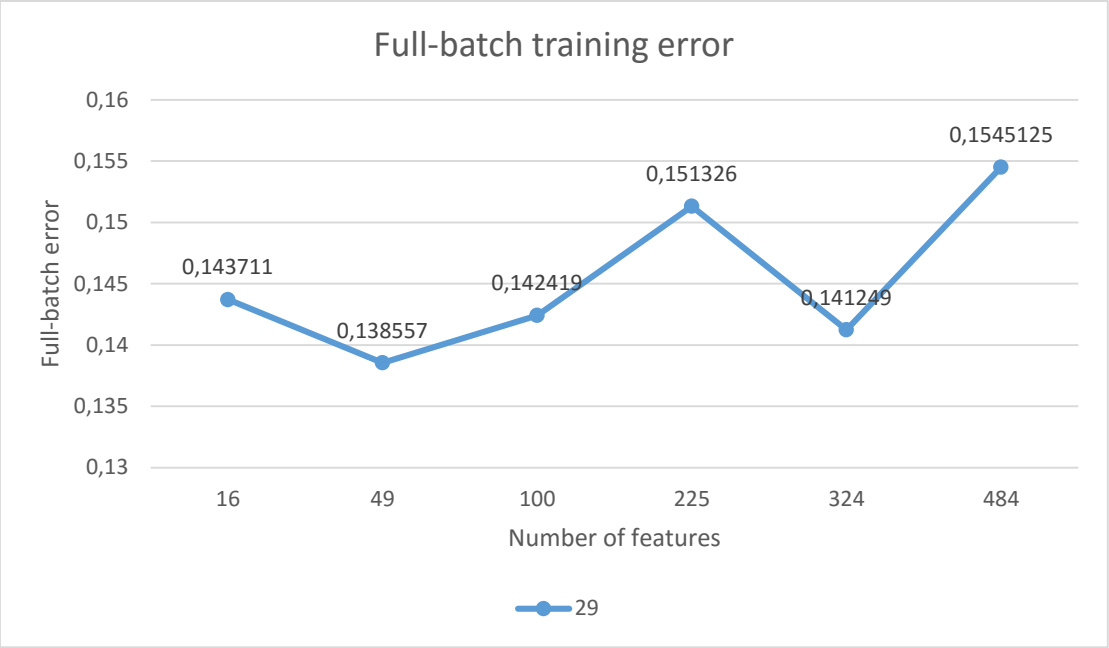


Figure 5-4: Features extracted: 1 epoch, 10 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

Next, the experiments are done with 1 epoch and 29 batchsize.





We can see that the mean squared error is minimum when the number of features is 100 and maximum when the number of features is 225. Likely, the full-batch error is minimum for 49 features and maximum for 225 features.

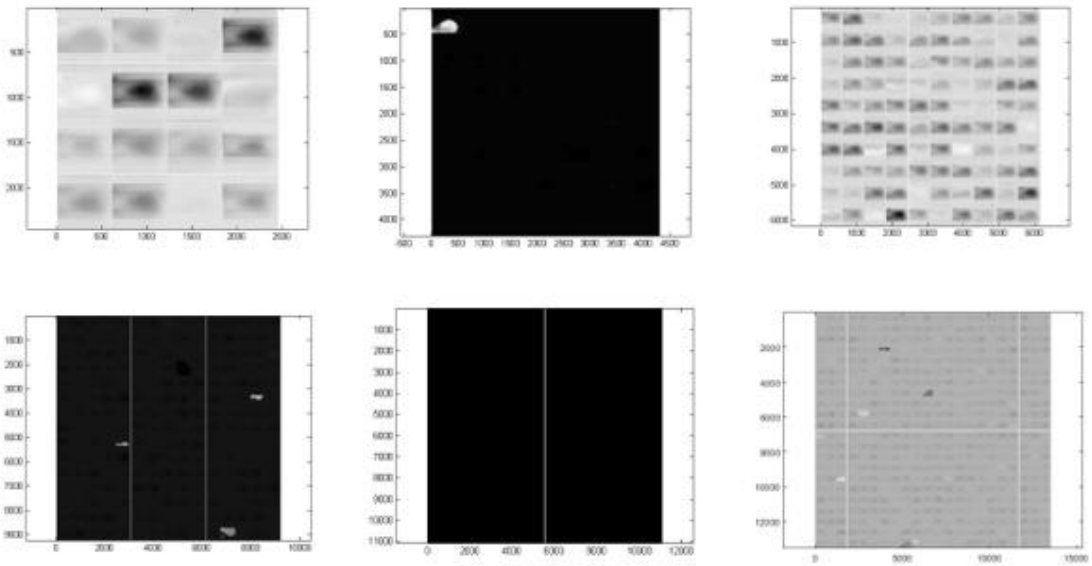
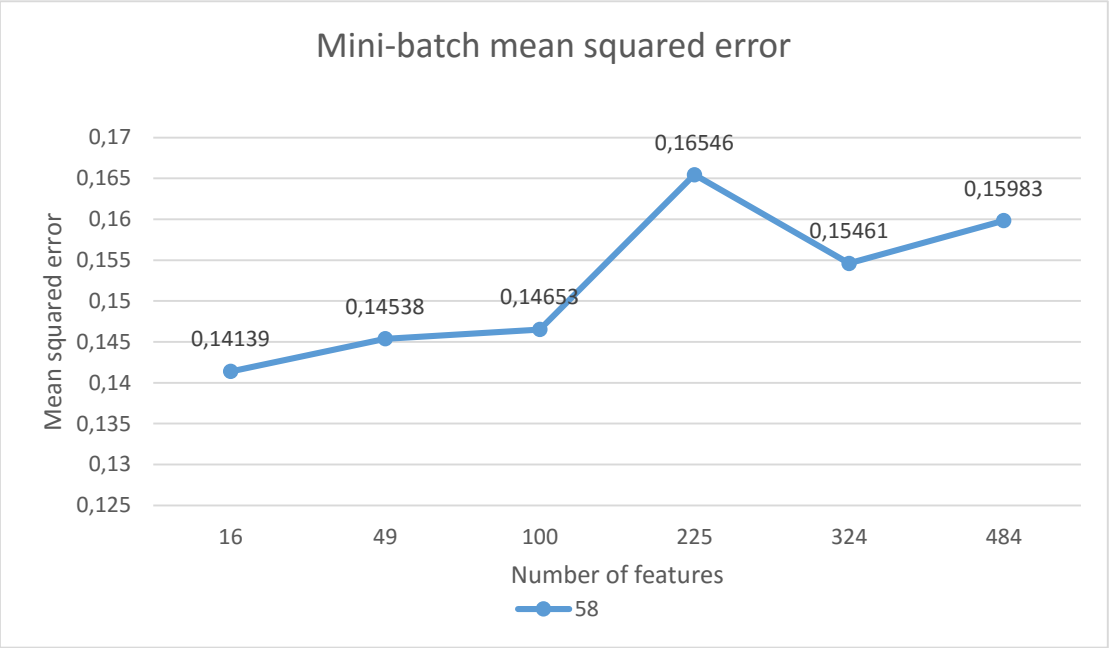
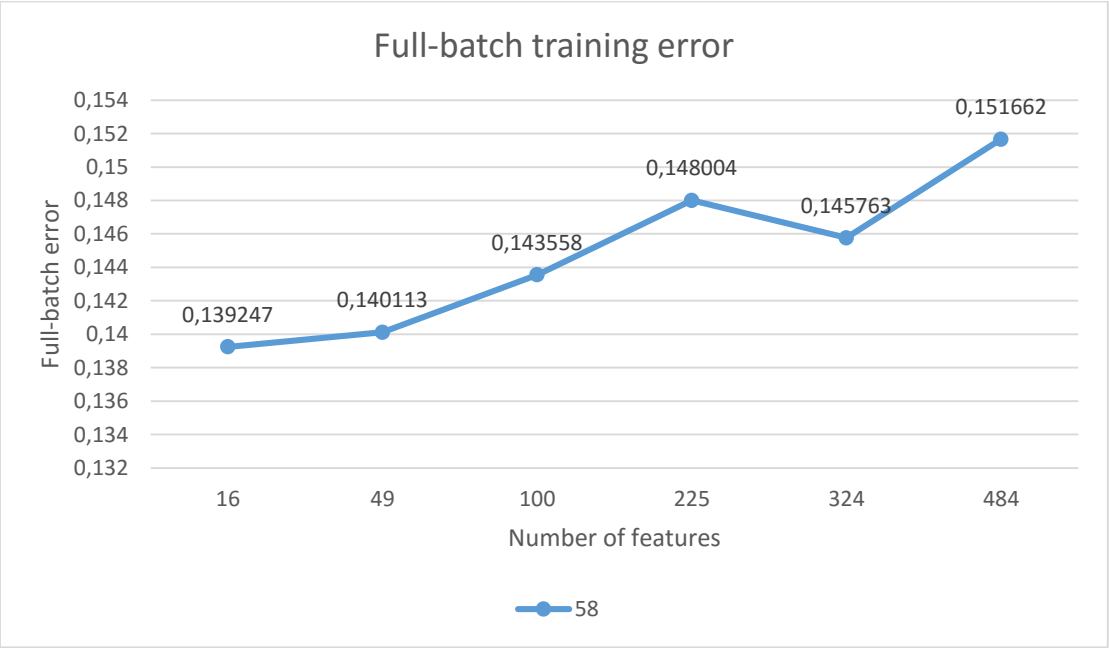


Figure 5-5: Features extracted: 1 epoch, 29 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

In the next experiment, we will work with 1 epoch, 58 batchsize and we will change the number of features.





We can see, again, that, as the number of features increases, so do both the mean squared error and the full-batch training error. The maximum errors are for 225 features and the minimum for 16 features.

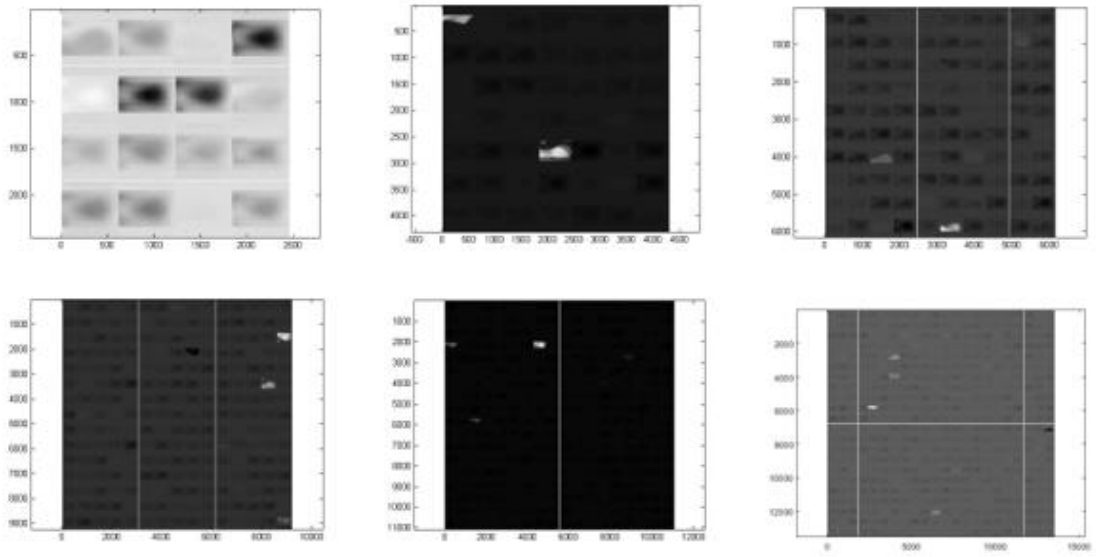
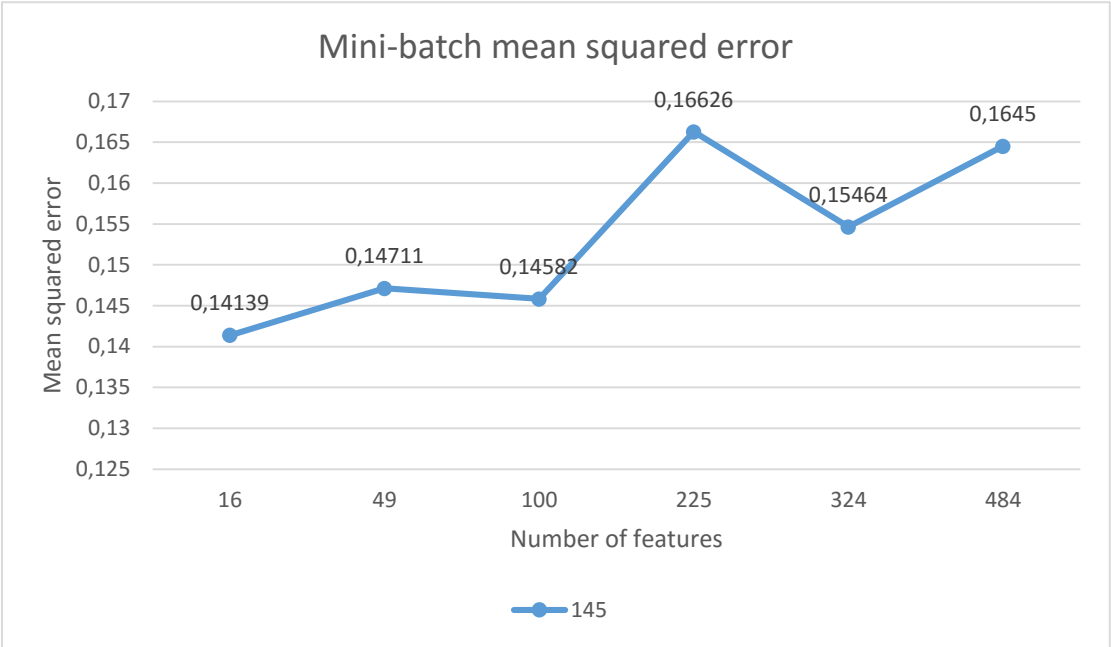
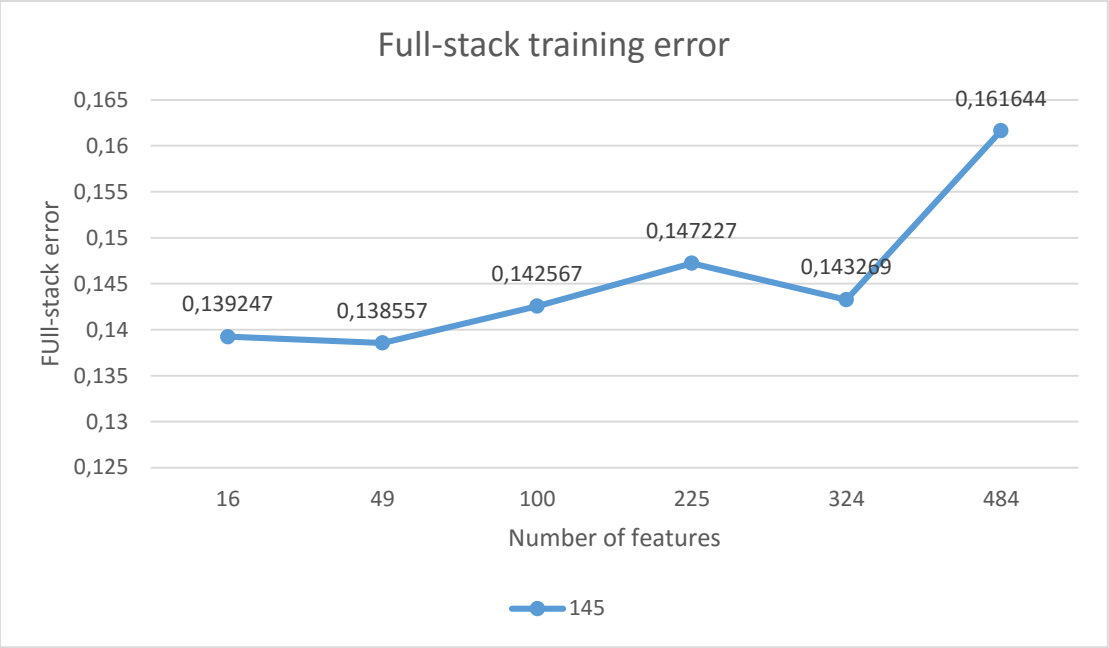


Figure 5-6: Features extracted: 1 epoch, 58 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

The next experiment was done using 1 epoch and 145 batchsize.





We can see that the mean squared error is maximum on 225 features, and is generally increasing with the increase of the number of the features. As far as the full-stack training error is concerned, it is also increasing with the increase of the number of features, with the exception of 324 features, where the error is smaller.

Finally, we experimented with 1 epoch and 290 batchsize.

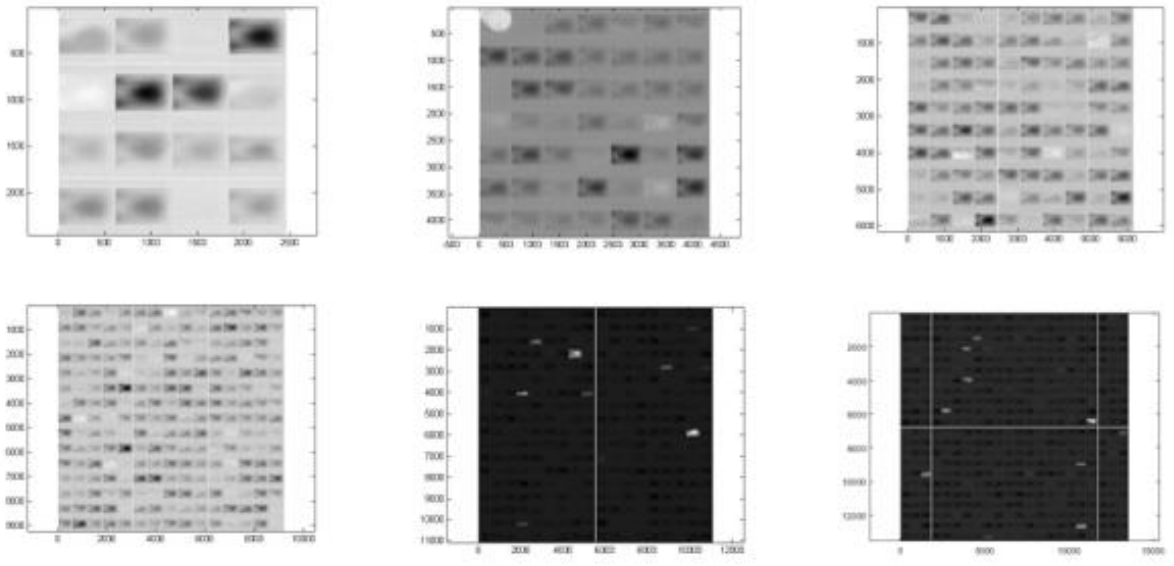
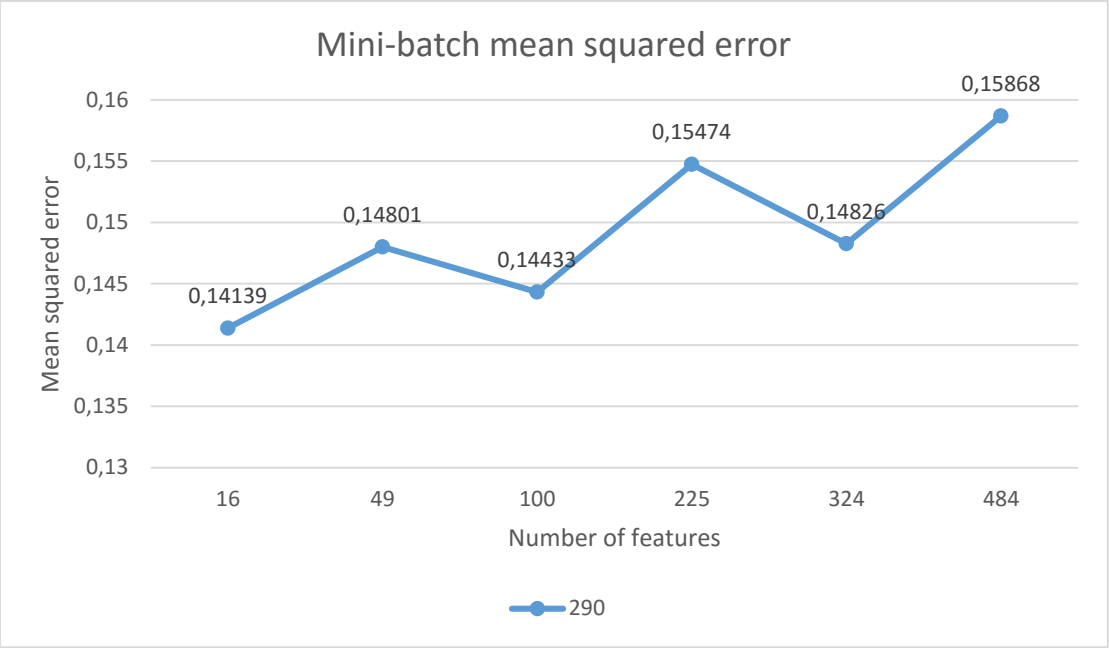
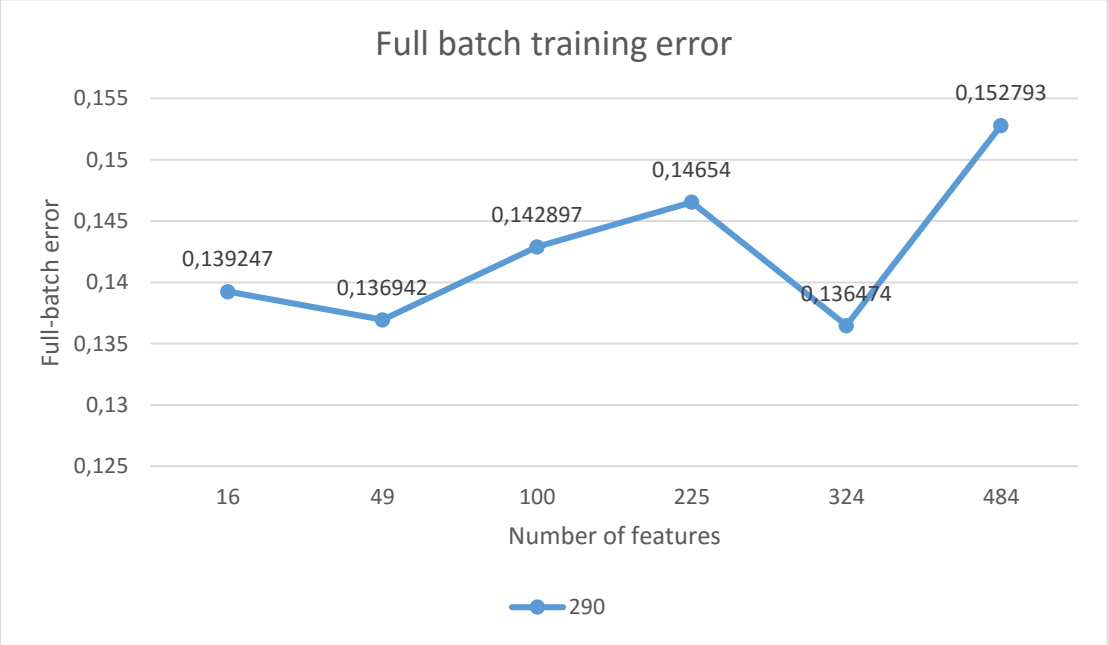


Figure 5-7: Features extracted: 1 epoch, 145 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





The mini-batch error is increasing as the number of features increases, but it is smaller on 100 and 324 features.

The full-batch error reduces from 16 to 49 features, then increases for 100 and 225, then it has the minimum value for 324 features, and the maximum for 484.

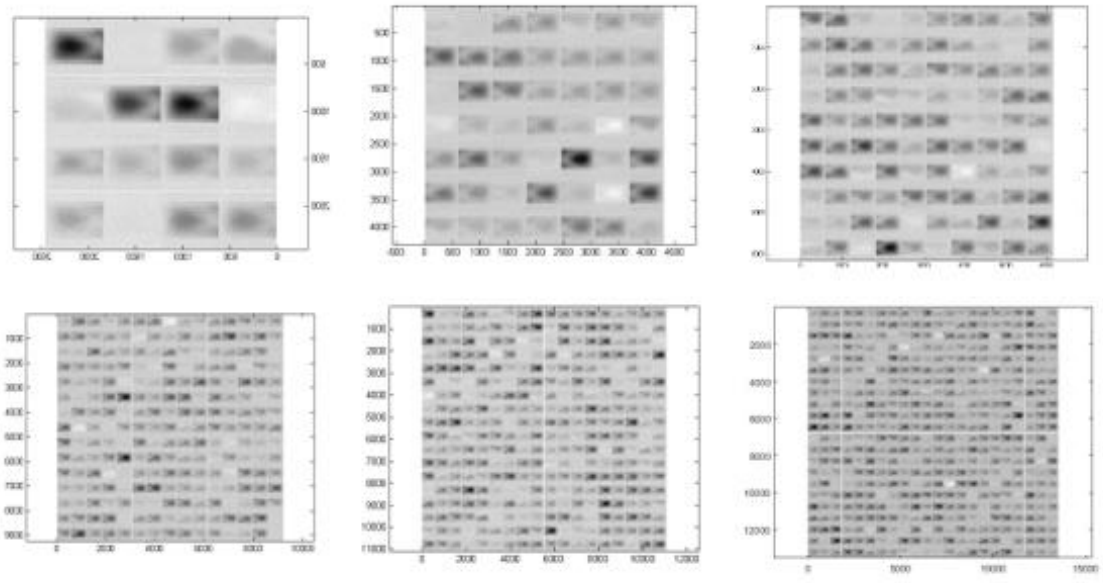


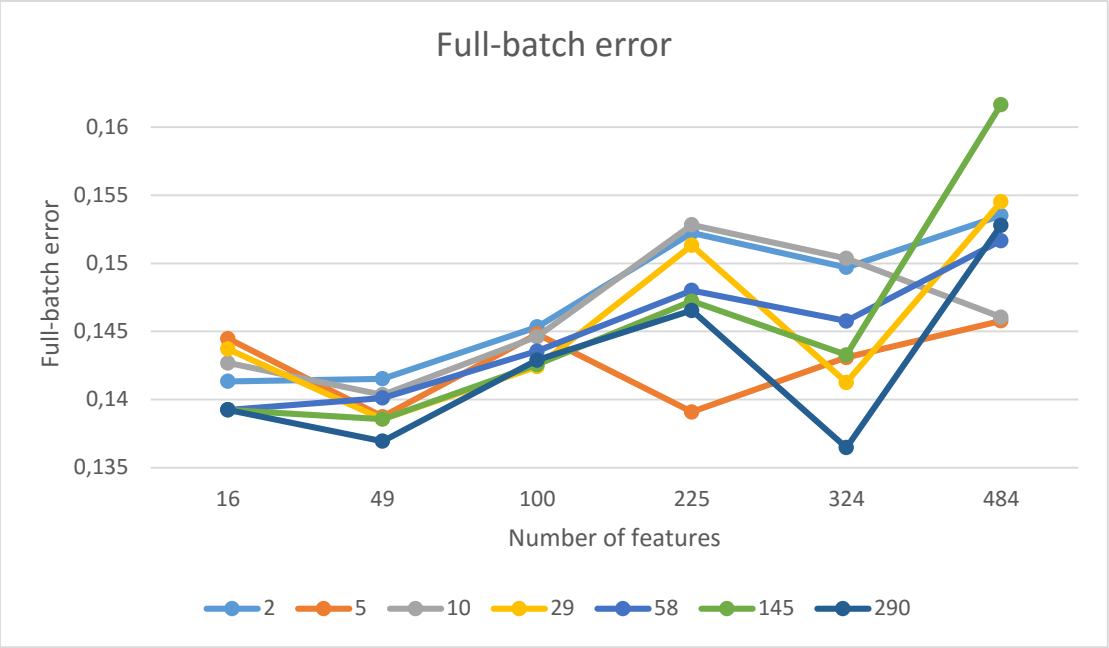
Figure 5-8: Features extracted: 1 epoch, 290 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

.

	2	5	10	29	58	145	290
16	0,141339	0,144467	0,142688	0,143711	0,139247	0,139247	0,139247
49	0,141525	0,138743	0,140347	0,138557	0,140113	0,138557	0,136942
100	0,145335	0,144818	0,14461	0,142419	0,143558	0,142567	0,142897
225	0,152244	0,139096	0,152827	0,151326	0,148004	0,147227	0,14654
324	0,149702	0,143087	0,150353	0,141249	0,145763	0,143269	0,136474
484	0,15351	0,145772	0,146042	0,154513	0,151662	0,161644	0,152793

Table 1: Full-batch error for experiments with 1 epoch.

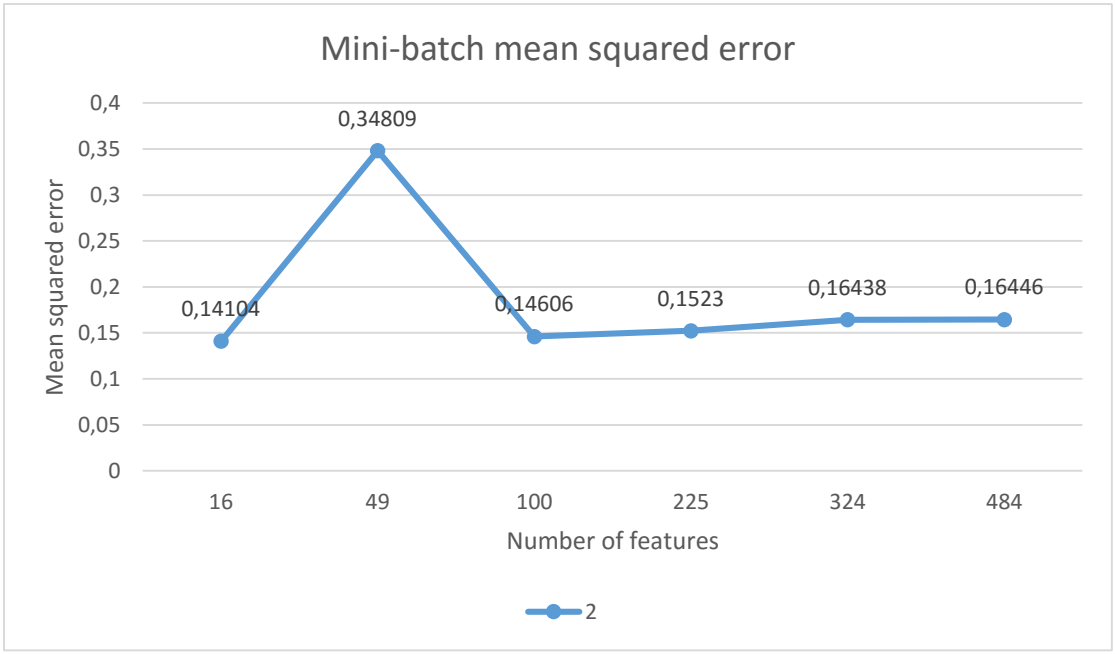
Putting all this data in a cumulative graph, we have:

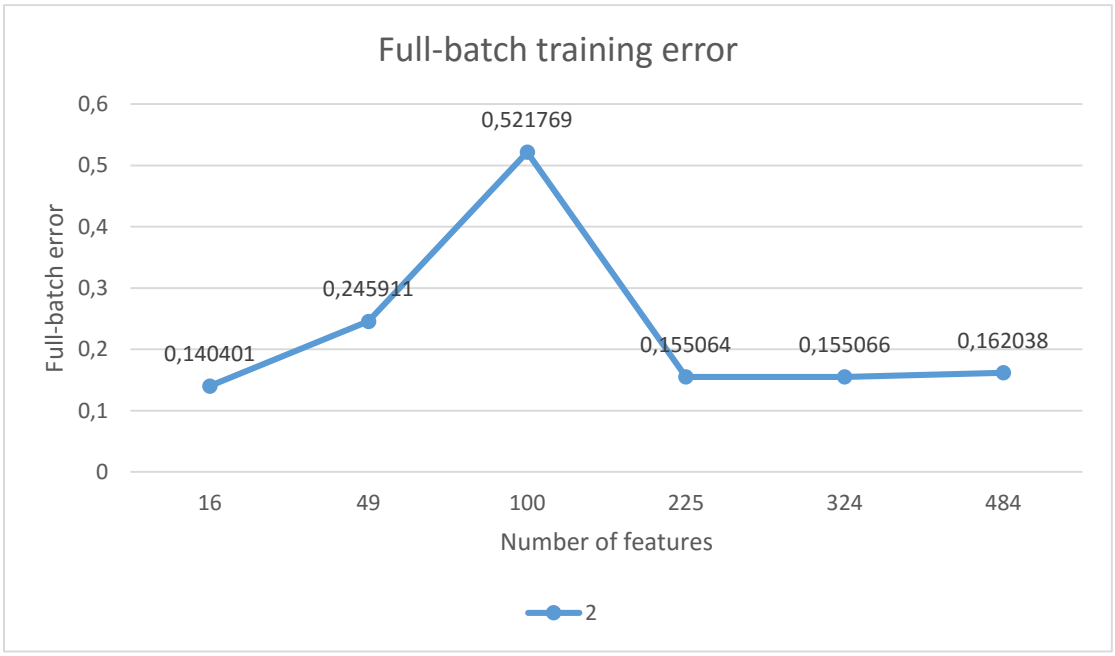


In this graph we see that as we start with a small number of features and a small batchsize, the error is small, and the smallest is with 58, 145 and 290 batchsize (0,139247). As the number of features and the batchsize increase, so does the error. In fact, when the number of features is 100, the error between different values of batchsize has very similar values, with a difference of just 0.002 (0.142897-0145335). For 225 features, the minimum error is for 5 batchsize (0.139096). As the number of features increases on 324, we can see a general decline and the value of the error decreases. Here we have the minimum error for the first set of experiments with 58 batchsize (0.136474). Finally, for 484 features, we have the maximum error for this set of experiments (0.161644).

4.2.2. Second set of experiments

The second set of experiments was done with 3 epochs, 2, 5, 10, 29, 58, 145 and 290 batchsize, and 16, 49, 100, 225, 324 and 484 features. We are now going to present the results in the form of graphs, as we did before, along with the images of the features extracted.


Mini-batch mean squared error
Mean squared error
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0,34809
0,14104
0,14606
0,1523
0,16438
0,16446
16
49
100
225
324
484
Number of features
2


Full-batch training error
Full-batch error
0,6
0,5
0,4
0,3
0,2
0,1
0
0,521769
0,245911
0,140401
0,155064
0,155066
0,162038
16
49
100
225
324
484
Number of features
2

The mini-batch mean squared error has the maximum value for 49 features, with a big distance from the other values. The full-batch error has the maximum value for 100 features, again with a big distance from the other values.

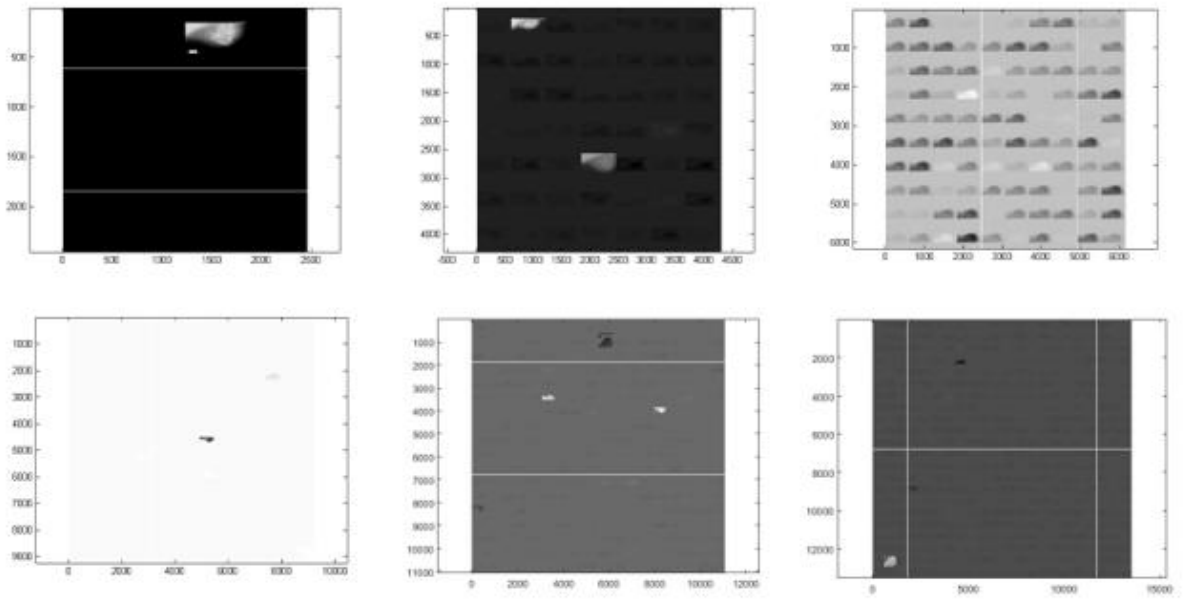
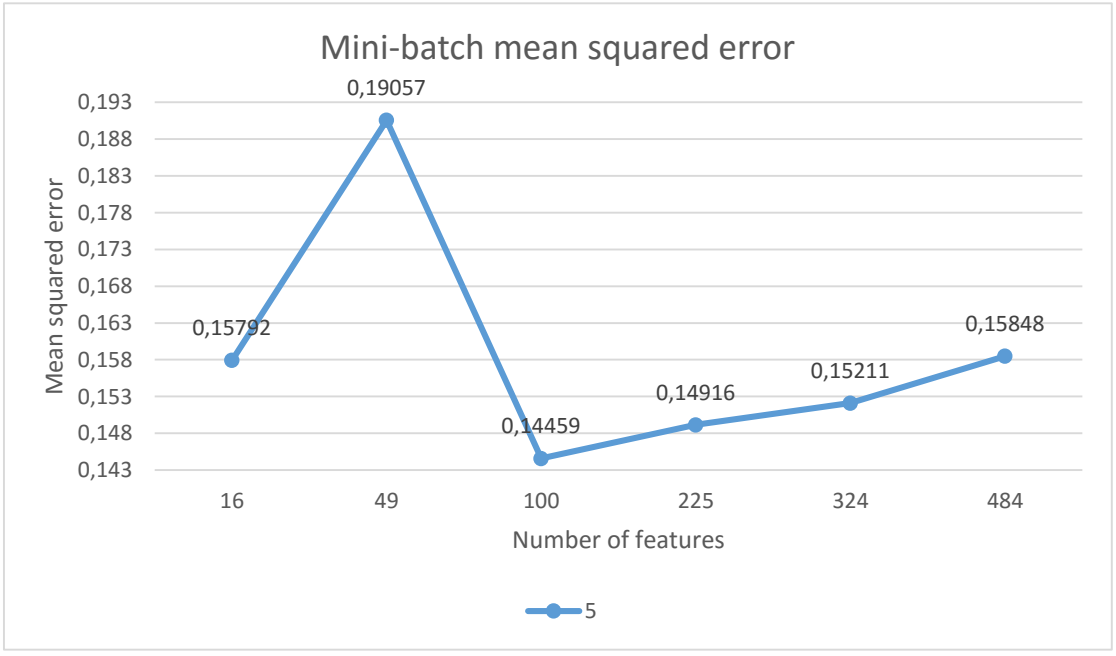
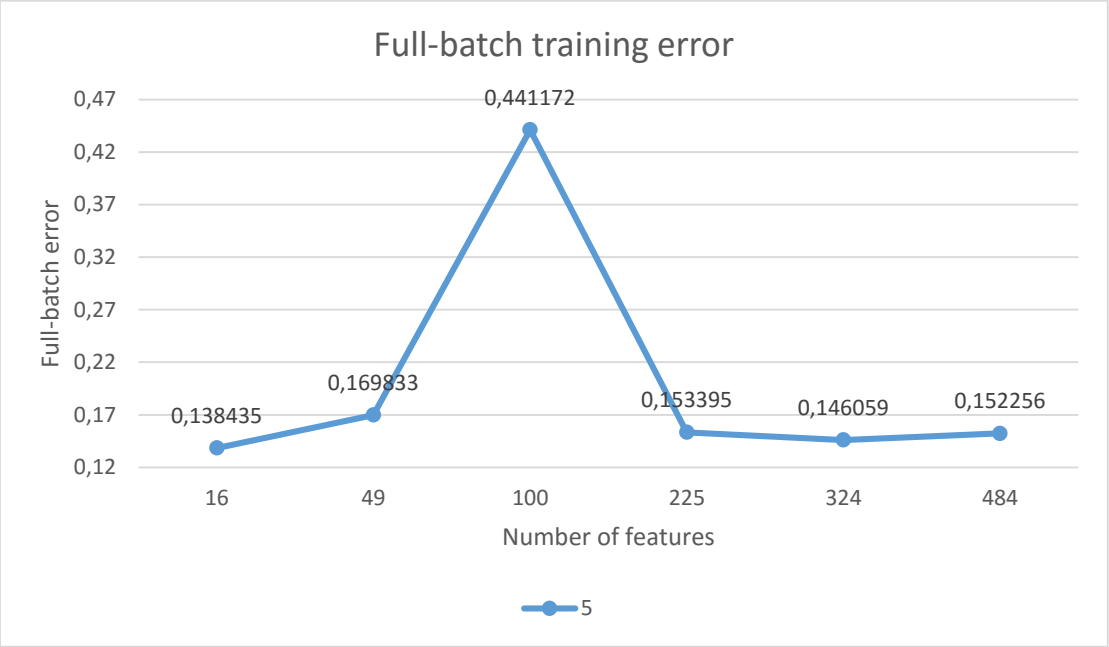


Figure 5-9: Features extracted: 3 epochs, 2 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





We can see that the mini-batch mean squared error has a sudden increase in its value (maximum value) for 49 features and then drops to its minimum value for 100 features. The maximum value of the full-batch training error is for 100 features, and its minimum for 16 features, though with a small distance from the value that it has with the remaining number of features.

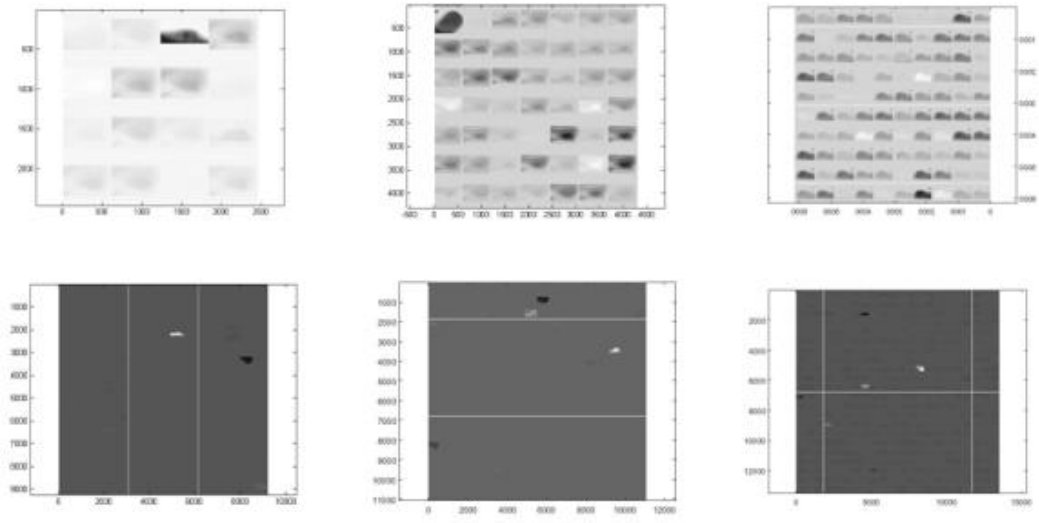
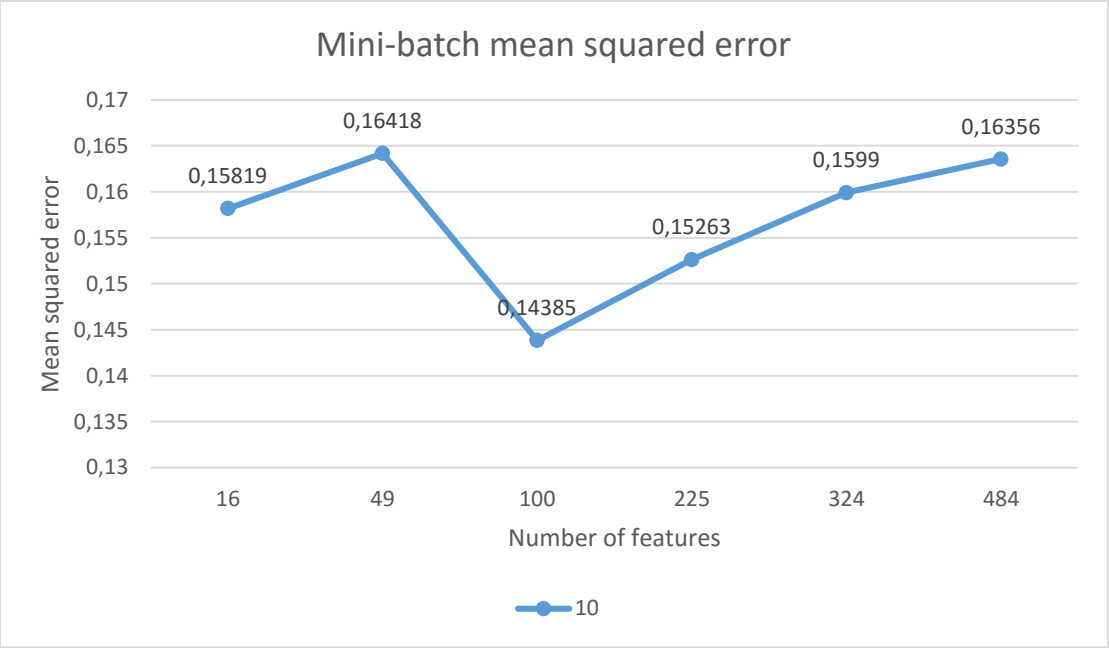
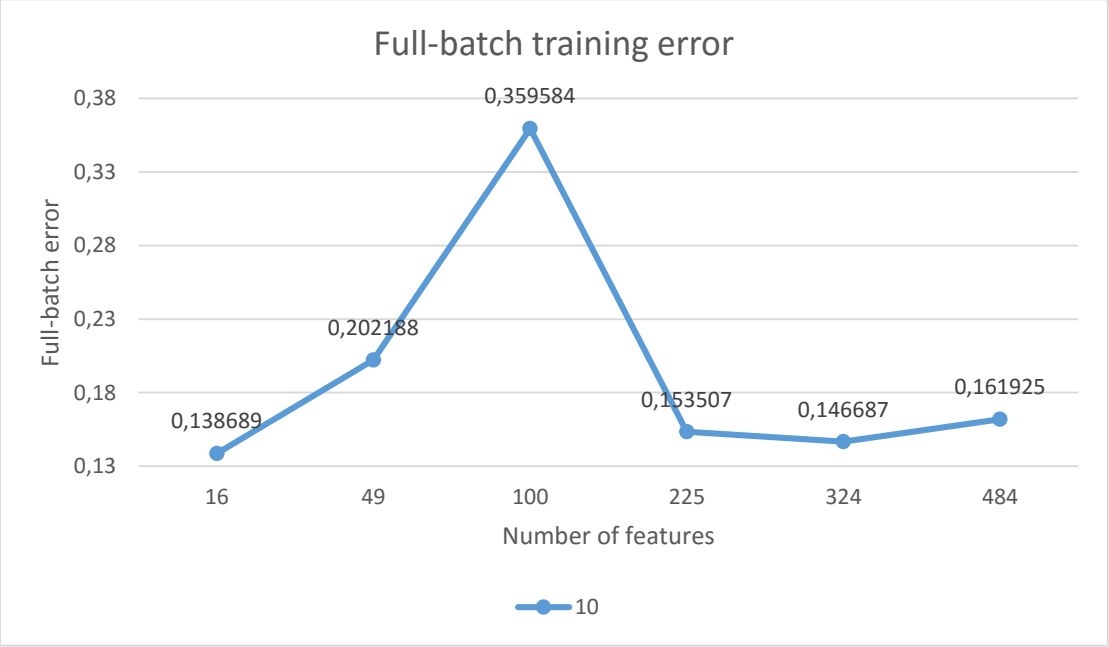


Figure 5-10: Features extracted: 3 epochs, 5 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





In the graph of the mini-batch mean squared error, we can see that the minimum value of the error is for 100 features and its maximum value for 49 features.

In the graph of the full-batch error, we can see that the maximum value of the error is for 100 features, and its minimum value for 16 features.

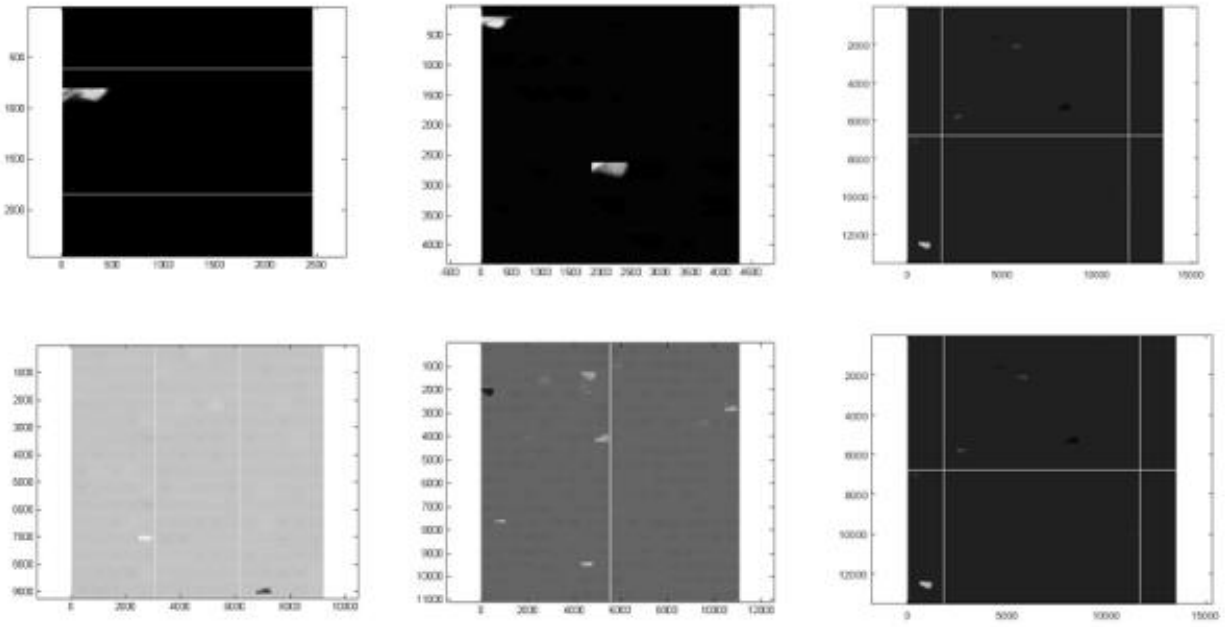
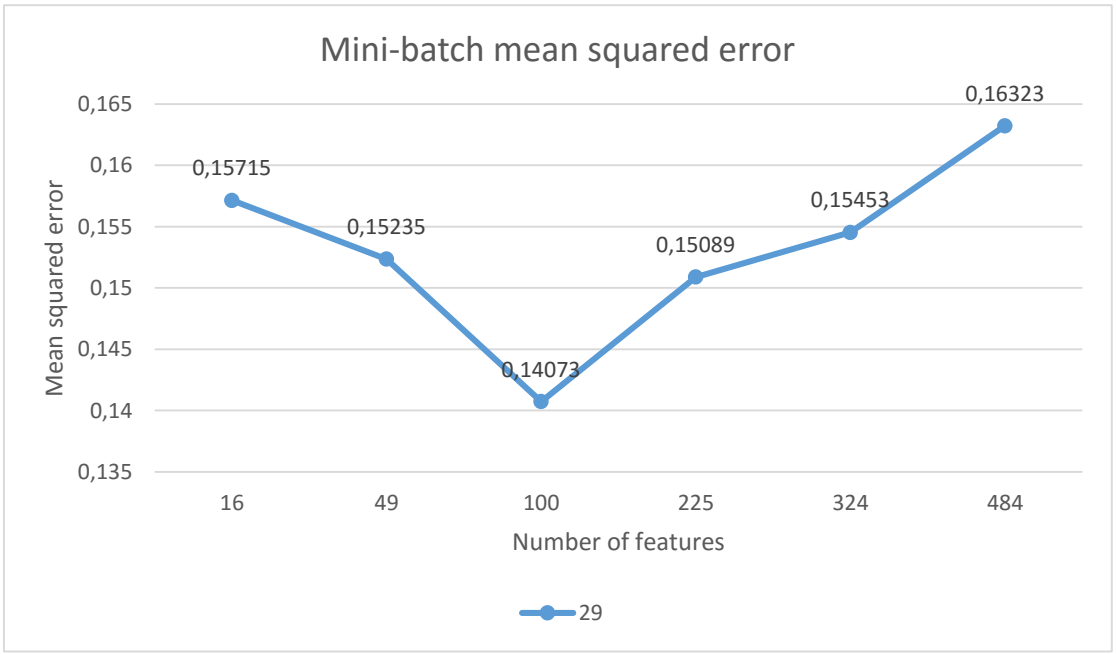
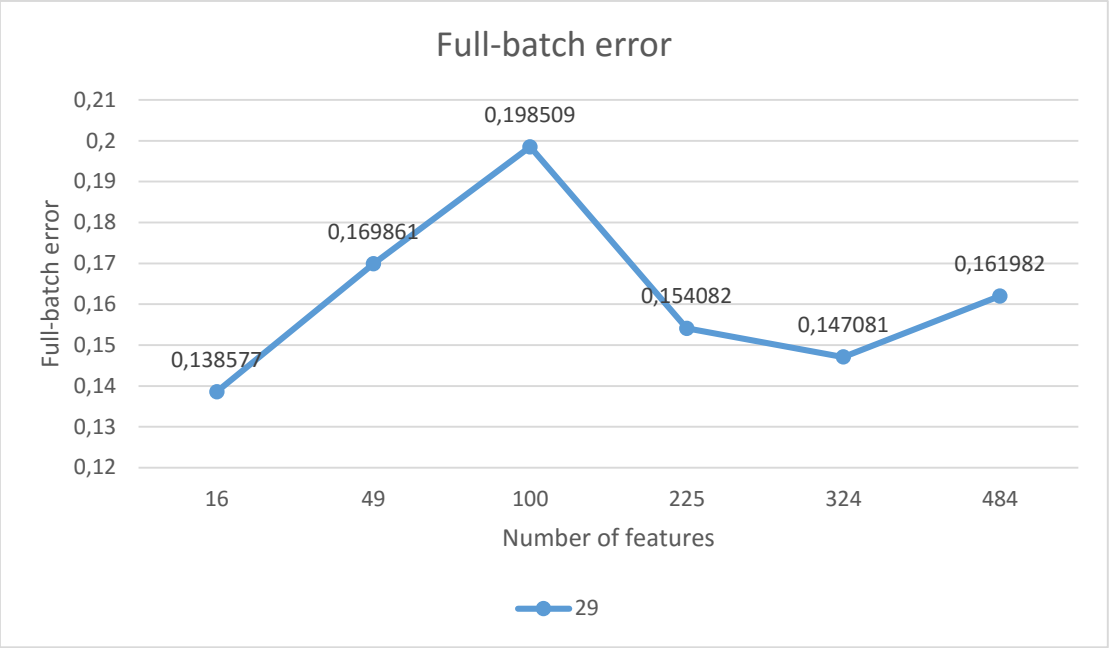


Figure 5-11: Features extracted: 3 epochs, 10 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





In the case of 29 batchsize, the value of the mini-batch error decreases when we go from 16 to 100 features, where it has its lowest value, and increases with the increase of the number of features to 484.

The value of the full-batch error increases with the increase of the number of features from 16 to 100, where it has its maximum value and then decreases.

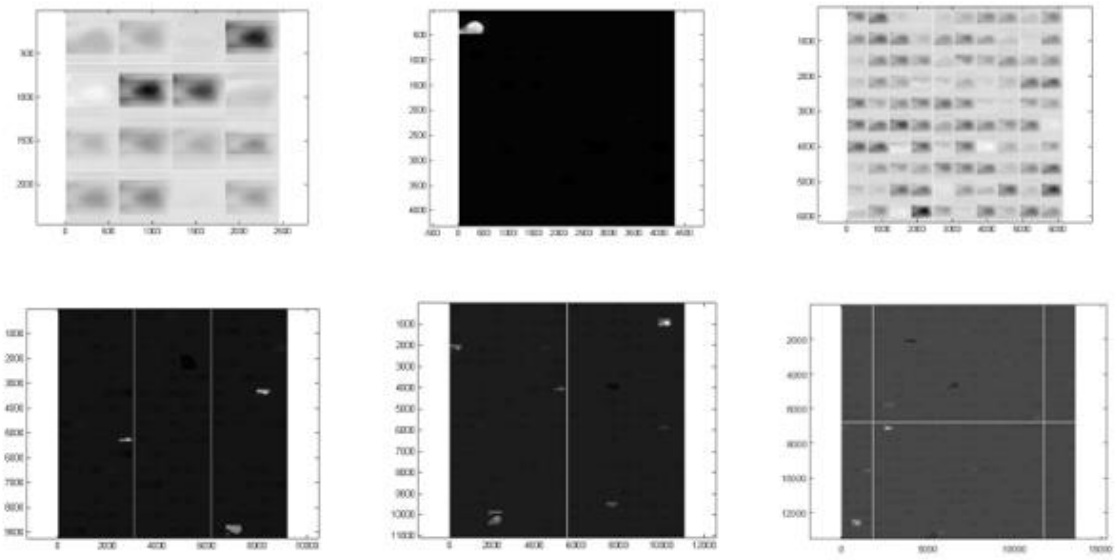
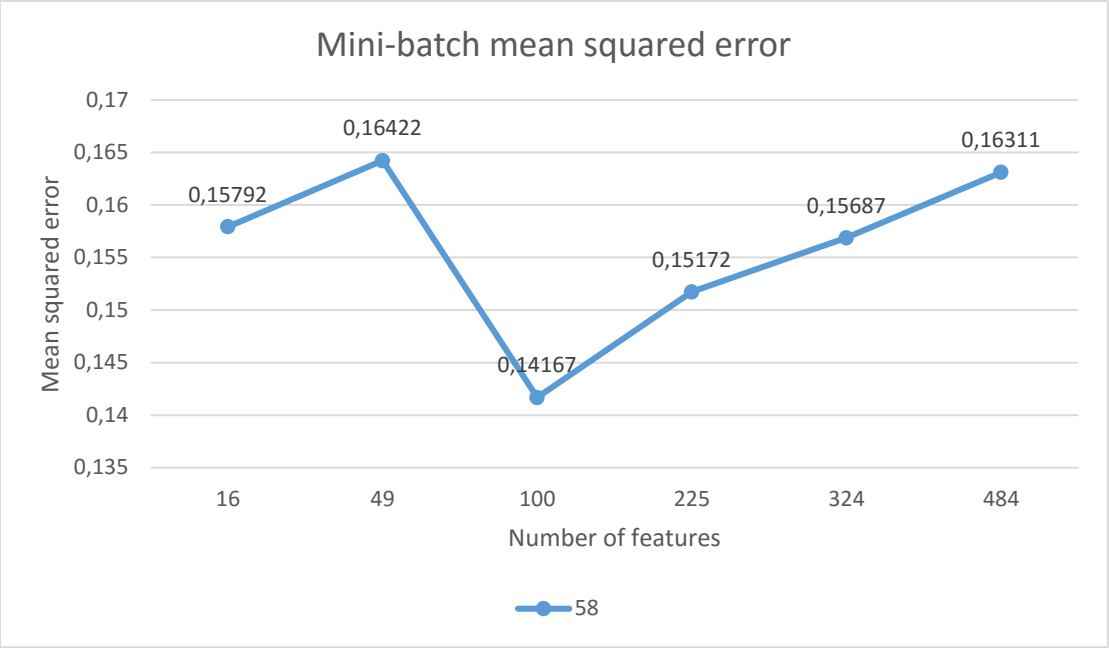
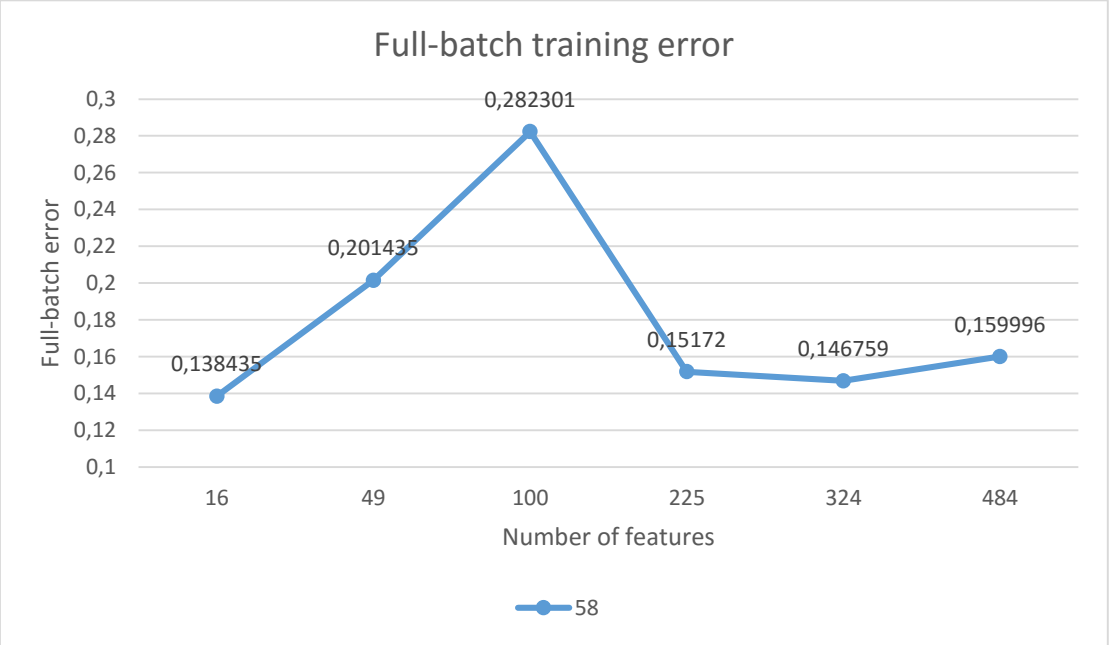


Figure 5-12: Features extracted: 3 epochs, 29 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





We can see again that the mini-batch error has its minimum value for 100 features, and the full-batch error for 16 features.

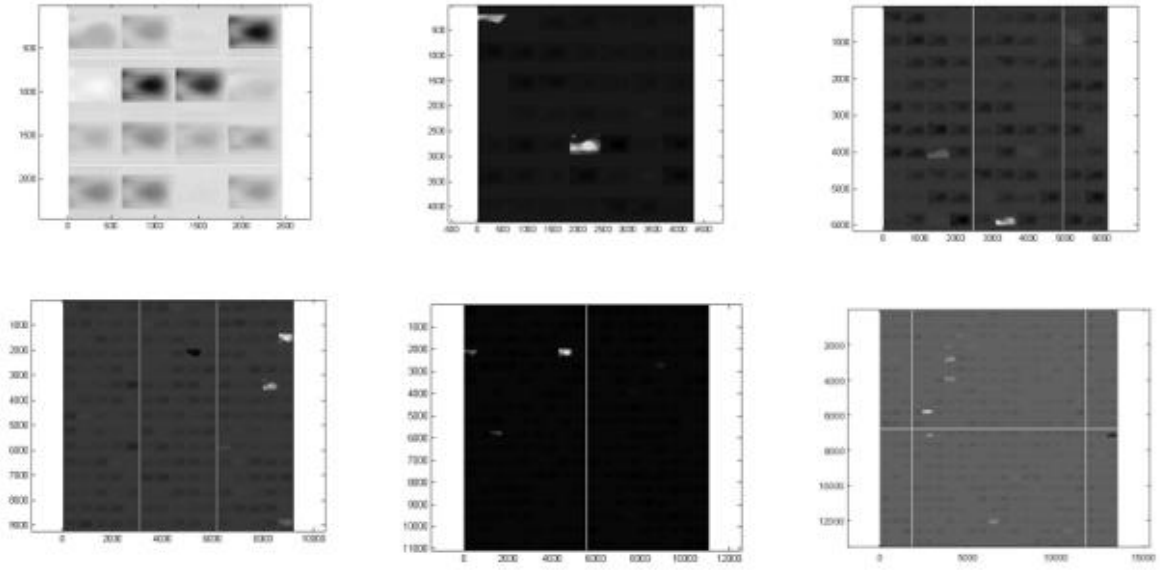
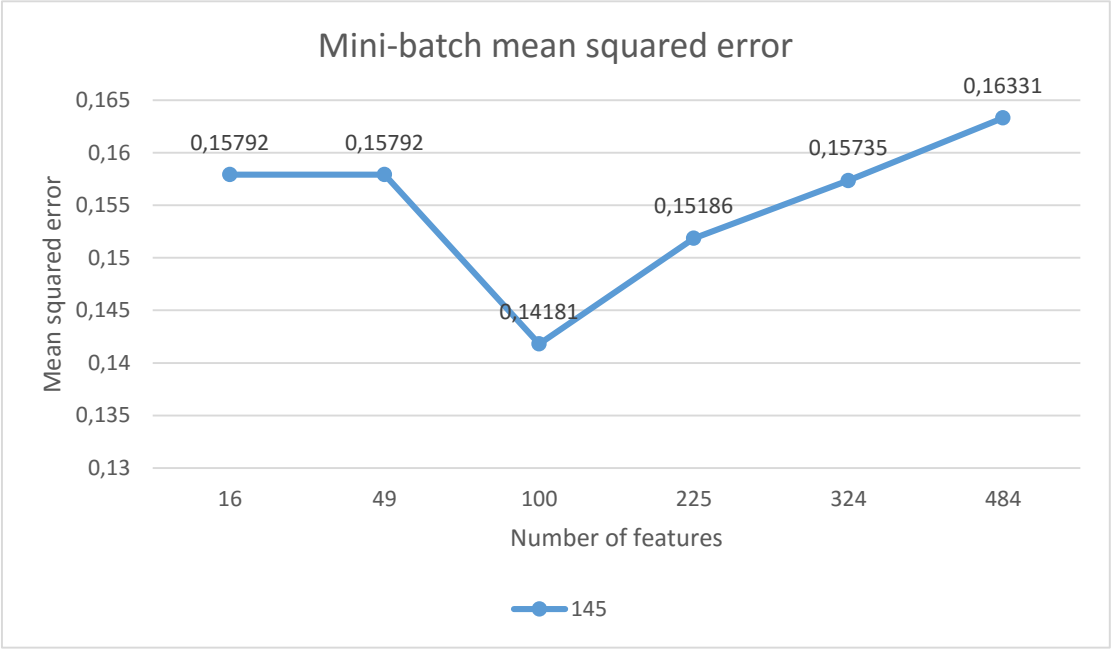
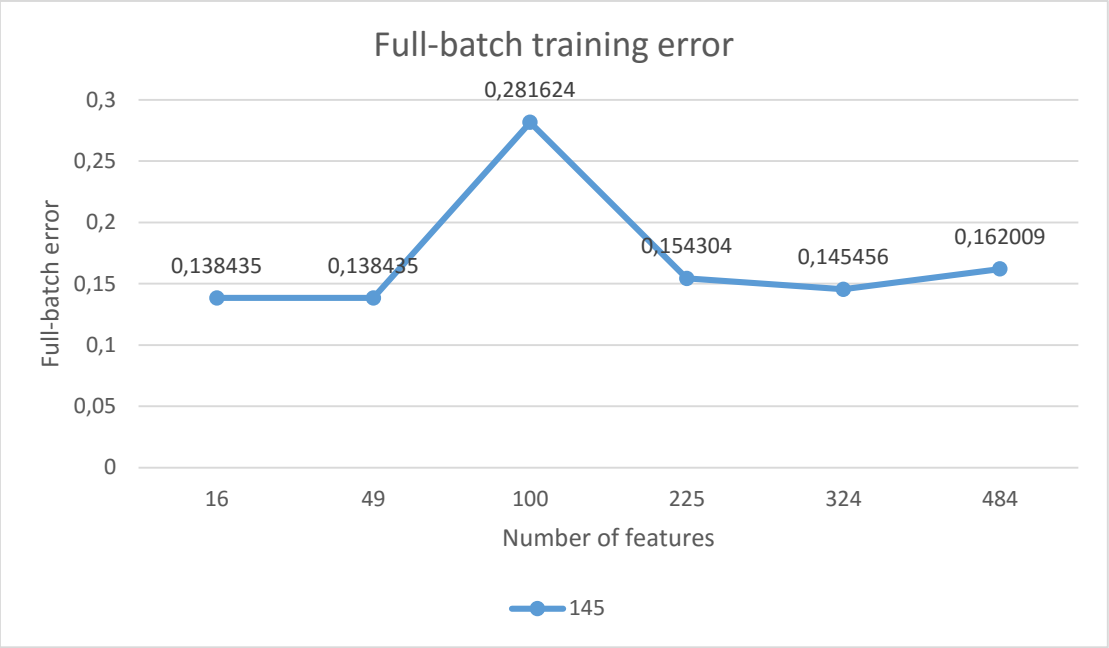


Figure 5-13: Features extracted: 3 epochs, 58 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).





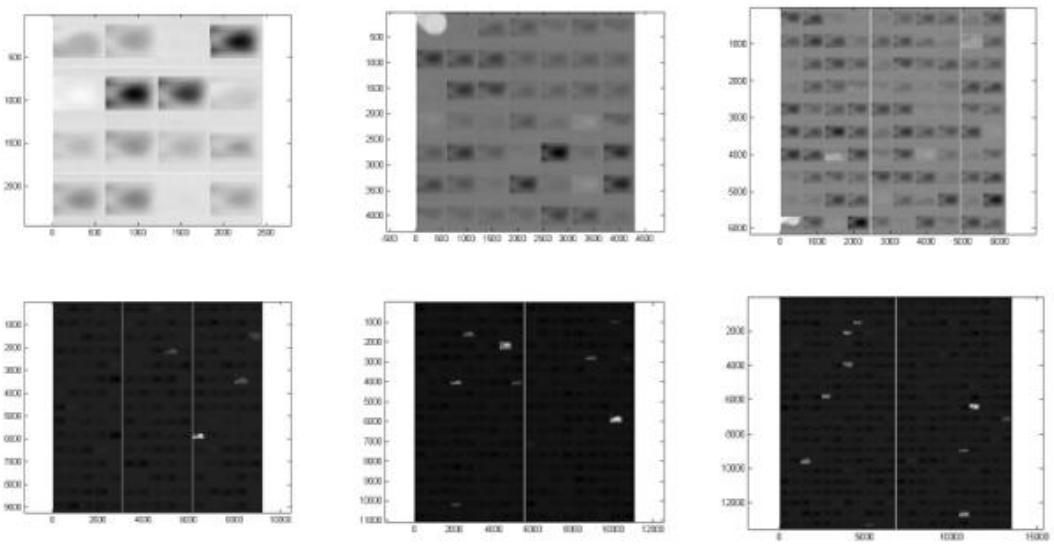
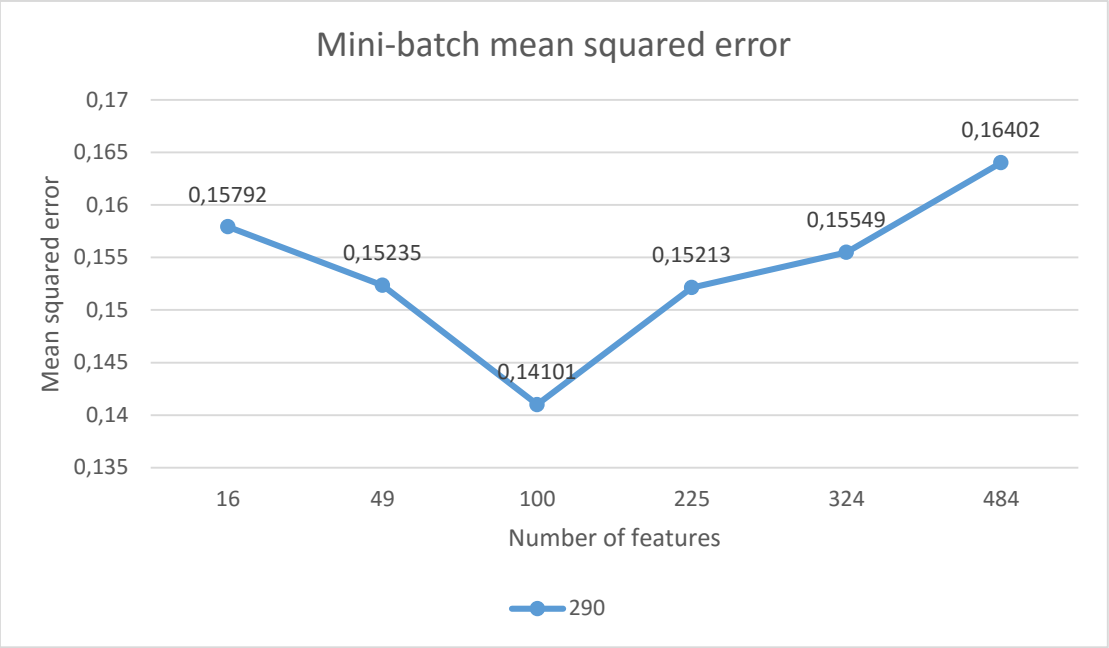
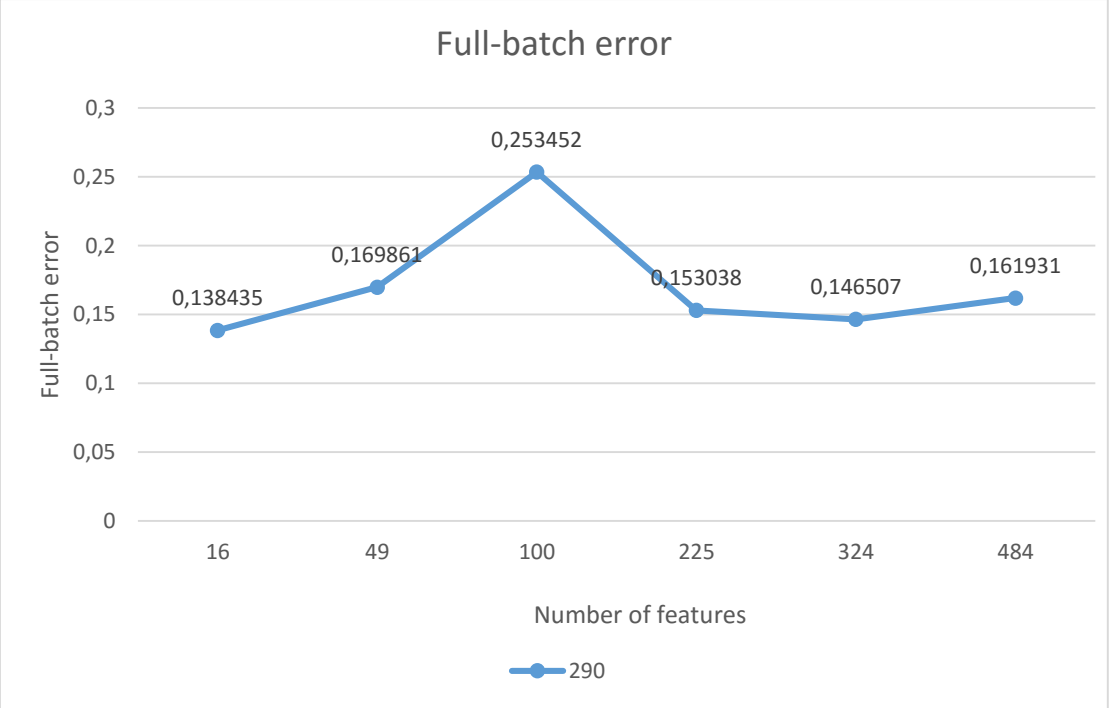


Figure 5-14: Features extracted: 3 epochs, 145 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).


Mini-batch mean squared error
Mean squared error
0,17
0,165
0,16
0,155
0,15
0,145
0,14
0,135
0,15792
0,15235
0,14101
0,15213
0,15549
0,16402
16
49
100
225
324
484
Number of features
290


Full-batch error
Full-batch error
0,3
0,25
0,2
0,15
0,1
0,05
0
0,138435
0,169861
0,253452
0,153038
0,146507
0,161931
16
49
100
225
324
484
Number of features
290

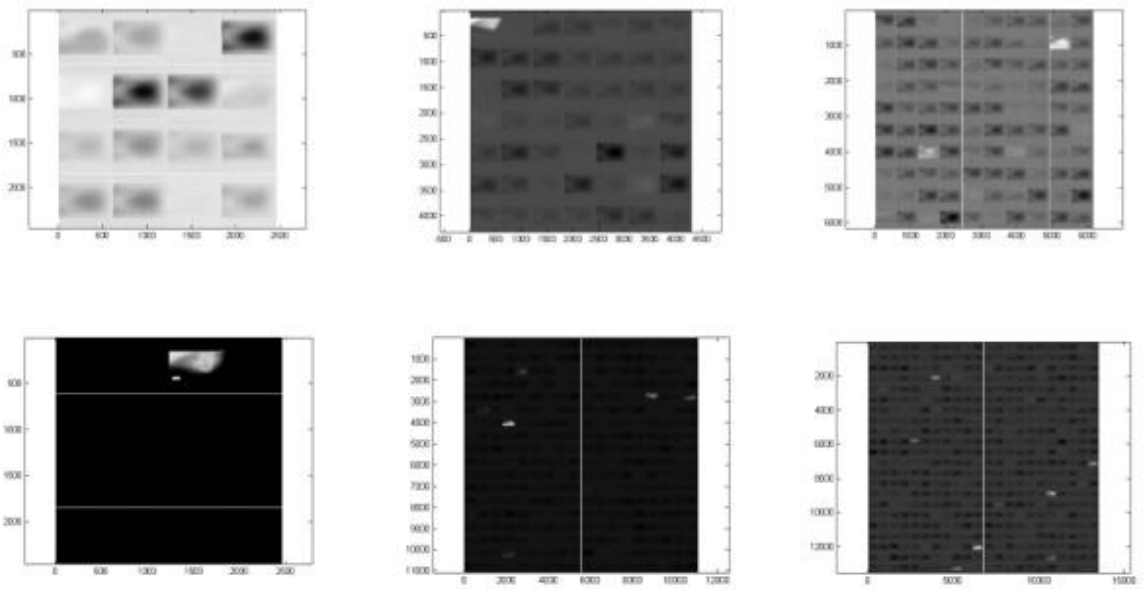
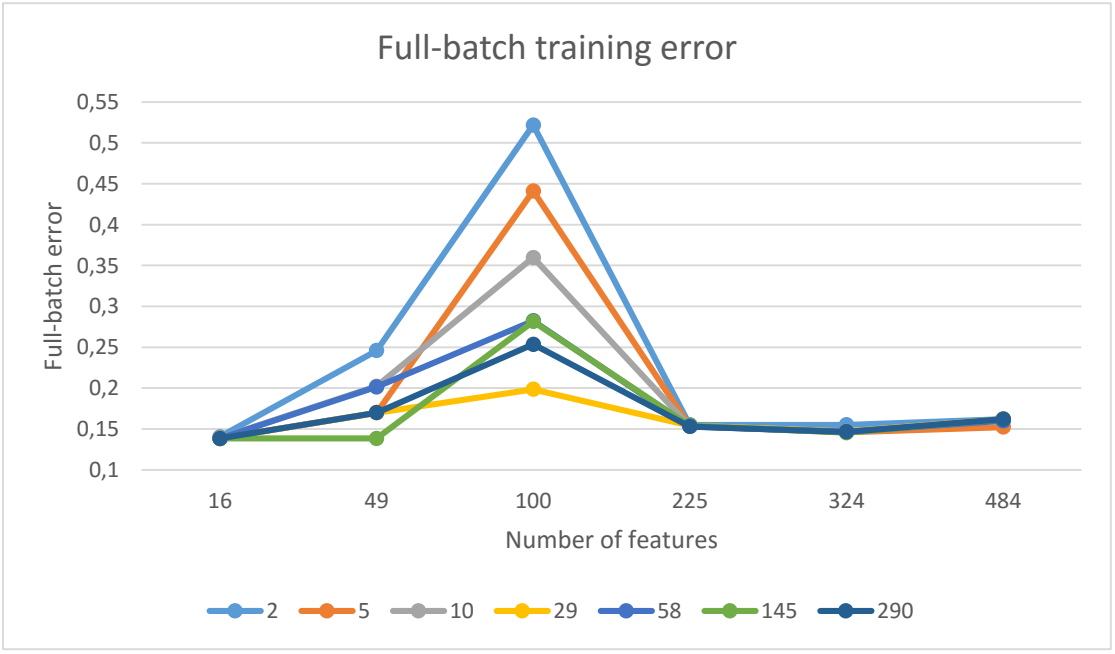


Figure 5-15: Features extracted: 3 epochs, 290 batchsize and 16, 49, 100, 225, 324, 484 features (left to right).

	2	5	10	29	58	145	290
16	0,140401	0,138435	0,138689	0,138577	0,138435	0,138435	0,138435
49	0,245911	0,169833	0,202188	0,169861	0,201435	0,138435	0,169861
100	0,521769	0,441172	0,359584	0,198509	0,282301	0,281624	0,253452
225	0,155064	0,153395	0,153507	0,154082	0,153033	0,154304	0,153038
324	0,155066	0,146059	0,146687	0,147081	0,146759	0,145456	0,146507
484	0,162038	0,152256	0,161925	0,161982	0,159996	0,162009	0,161931

Table 2: Full-batch error for experiments with 3 epochs.

We collected all this data into a graph:



In this graph we can see that, for all values of batchsize, when the number of features is 100, the value of the full-batch error maximizes. The maximum error of the second set of experiments is for 100 features, 58 value of batchsize and 3 epochs (0.521769). The minimum error is for 16 and 49 features, with value of batchsize being 5, 58, 145, 290 and 3 epochs (0.138435). As the number of the features increases up to 100, the full-batch error increases too. Then it decreases and its value is maintained to low levels. We can also see that we have better results (smaller error) as the number of batchsize increases, with the exception of 145 and 290 batchsize.

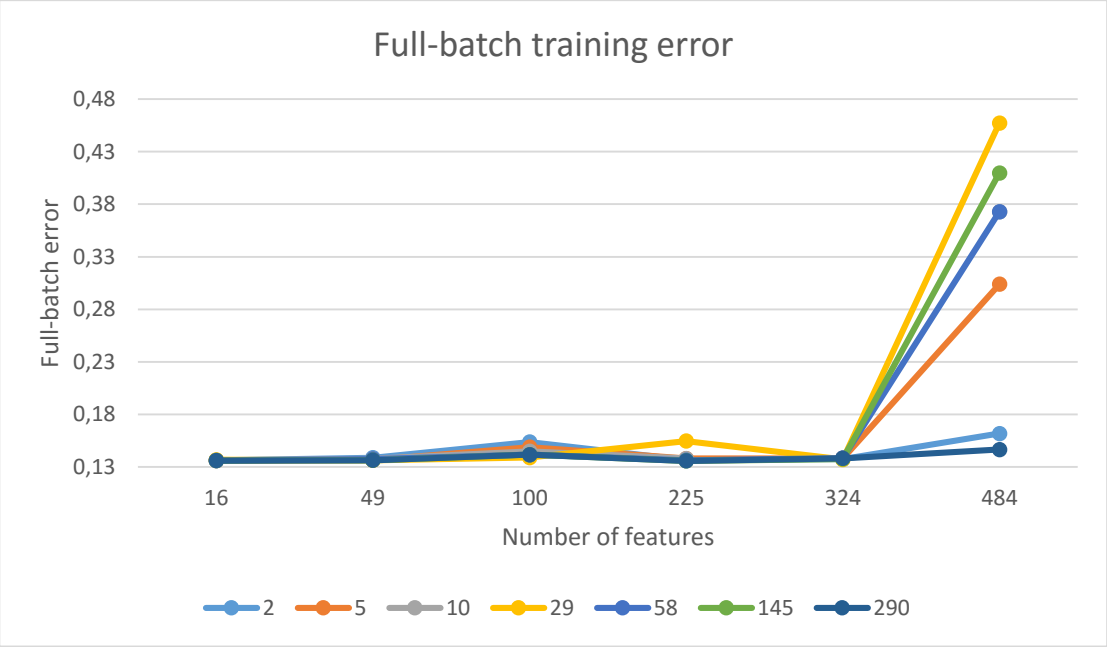
4.2.3. Third set of experiments

The third and last set of experiments was done trying 5 epochs, 2, 5, 10, 29, 58, 145, 290 number of batchsize and 16, 49, 100, 225, 324, 484 features. We will now present the results in the following table:

	2	**5**	**10**	**29**	**58**	**145**	**290**
16	0.136276	0.136108	0.136298	0.136897	0.136108	0.136108	0.136108
49	0.138820	0.136658	0.137730	0.136404	0.137618	0.136404	0.136404
100	0.153767	0.148932	0.144901	0.138973	0.141670	0.141670	0.141670
225	0.136558	0.137988	0.138108	0.154627	0.136351	0.135775	0.135801
324	0.137466	0.138133	-	0.137200	0.138555	0.137662	0.138204
484	0.161863	0.303840	0.372895	0.457262	0.372895	0.409746	0.146554

Table 3: Full-batch error for experiments with 5 epochs.

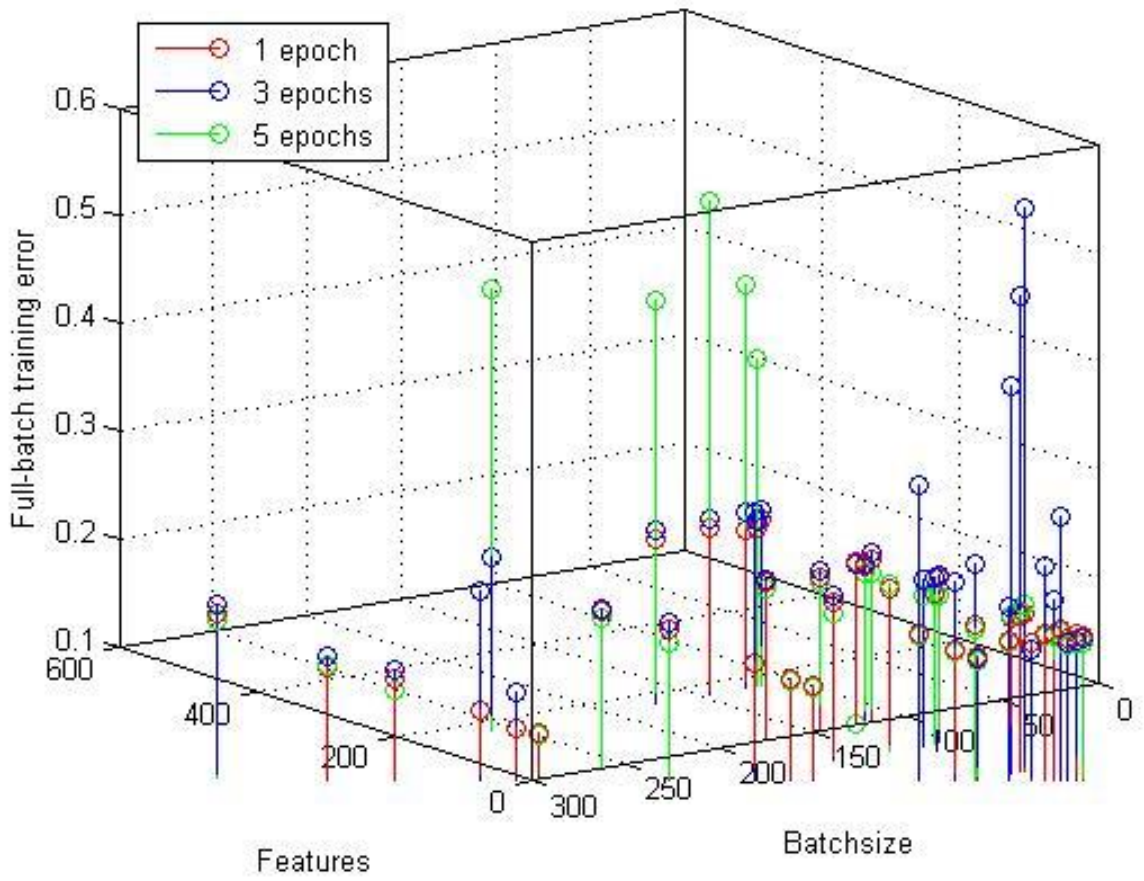
Putting this data into a graph, we have:

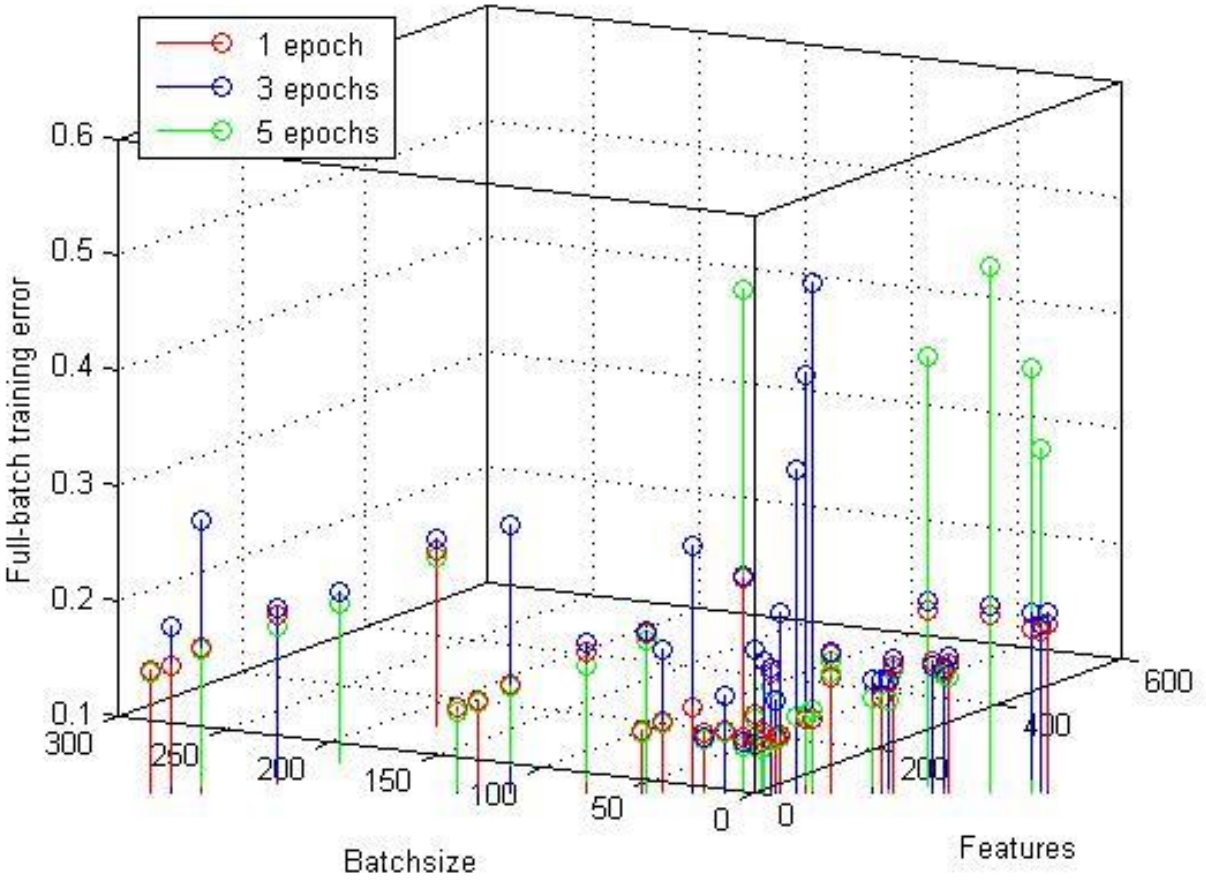


«Breast cancer classification of mammographies using stacked denoising autoencoders».

In this graph we can see that the minimum full-batch training error for this set of experiments with 5 epochs is for 225 features and 145 batchsize (0.135775). As the number of features increases, the value of the full-batch error stays at the same levels, except when the number of features is 100, where it increases a bit, and when the number of features is 484, where it reaches its maximum value for 29 batchsize (0.457262).

We made a 4d comparison graph in MATLAB in order to see how each of the parameters we are experimenting on (epochs, batchsize, features) affects the full-batch training error.





In this graph, the x axis is the number of features, the y axis the number of batchsize and the z axis the full-batch training error. We can see that the highest values of the error occur for 3 epochs, when the number of features is relatively small to medium and the number of batchsize small, and for 5 epochs, when the number of features is relatively big and the number of batchsize small to medium. The maximum value of the full-batch training error of all the sets of experiments occurs for 3 epochs, 100 features and 2 batchsize (0.521769). The value of the error is also high for 3 epochs, 100 features and 5 batchsize (0.441172), for 5 epochs, 484 features and 10 batchsize (0.372895), for 3 epochs, 100 features, 10 batchsize (0.359584), for 5 epochs, 145 batchsize and 484 features (0.409746) etc. The minimum value of the error of all the sets of experiments is for 5 epochs, 145 batchsize and 225 features (0.135775). The rest of the values of the error are relatively small and do not have big variations.

CHAPTER 5

5 CONCLUSIONS – FUTURE WORK

5.1. Conclusions

In this paper, we introduced the theory of machine learning. We gave two definitions for Machine Learning, described supervised and unsupervised learning and gave some examples. Next, the term of classification was introduced, along with the definitions of classes and classifiers.

In the next sections, we gave the problem definition and described the objects that can be found in a mammography that indicate the presence of breast cancer. We analyzed the theory of stacked denoising autoencoders and described the process of greedy layer-wise training.

Next, we trained a stacked denoising autoencoder and experimented on changing the values of specific parameters (epochs, batchsize and number of features). For each combination of the parameters and feature extraction, we compared the full-batch training error. Generally, the results were better for 5 epochs for smaller values of batchsize. However, we had equally good results for batchsize=290. Regarding the number of features, the error was maximum for 484 features. With 3 epochs, the results were better for medium values of batchsize and specifically for batchsize=29. The error maximized for each experiment when the number of features was 100. With 1 epoch, the error was generally maintained on low levels, and we did not notice any specific relation between the features, batchsize and the full-batch training error.

5.2. Future work

As seen from the experimental results, the process of feature extraction, using stacked denoising autoencoders, is satisfying. In future work, preprocessing will be used for the images, in order to remove redundant information and noise, so as to have even better results. Also, it would be suggested to perform more experiments, changing the values of other parameters, too, e.g. number of autoencoders, learning rate, batchsize and epochs of the classifier etc. So that the research is more thorough and better classification is achieved.

REFERENCES

[1] Zoubin Ghahramani (2004), Unsupervised Learning, Springer Berlin Heidelberg

[2] Alex Smola, S.V.N Vishwanathan (2008), Introduction to Machine Learning, Cambridge University Press

[3] "Machine Learning." Wikipedia:The Free Encyclopedia. Wikimedia Foundation, Inc. 25th February 2016. Web. https://en.wikipedia.org/wiki/Machine_learning

[4] Pedro Domingos (2012), A few useful things to know about machine learning, Communications of the ACM, 55, p. 78-87

[5] Christopher M. Bishop (2006), Pattern recognition and machine learning, Springer

[6] Cai E., (2014), Machine Learning Lesson of the day – Overfitting and Underfitting, https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting

[7] "Curse of dimensionality." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 12th February 2016. Web. https://en.wikipedia.org/wiki/Curse_of_dimensionality#Machine_learning

[8] Sammut, Claude, and Geoffrey I. Webb. Encyclopedia of machine learning. Springer Science & Business Media, 2011.

[9] Machine Learning: What it is and why it matters, (n.d.), http://www.sas.com/en_us/insights/analytics/machine-learning.html

[10] Michie, Donald, David J. Spiegelhalter, and Charles C. Taylor. "Machine learning, neural and statistical classification." (1994).

[11] Hala Al-Shamlan & Ali El-Zaart, (2010). Feature Extraction Values for Breast Cancer Mammography Images, Bioinformatics and Biomedical Technology (ICBBT), 2010 International Conference on, 335 - 340.

[12] Belal K. Elfarra1 & Ibrahim S. I. Abuhaiba2, (2013). New Feature Extraction Method for Mammogram Computer Aided Diagnosis, International Journal of Signal Processing, Image Processing and Pattern Recognition, 6 (1).

[13] American Cancer Society. Breast Cancer Facts & Figures 2013-2014. Atlanta: American Cancer Society, Inc. 2013.

[14] A. Wrublezca*, P. Bonizski, A. Przelaskowski & M. Kazubec, (2003). Segmentation and feature extraction for reliable classification of microcalcifications in digital mammograms, Opto-Electronics Review 11(3), 227–235

[15] Ryszard S. Chora, (2008). Shape and Texture Feature Extraction for Retrieval Mammogram in Databases in Information Technologies in Biomedicine, II, 121-128, Springer Berlin Heidelberg

[16] Halls S. (n.d.), Moose and Doc: Breast Cancer, http://breast-cancer.ca/abnorm-mams/

[17] The mini-MIAS Database of Mammograms. http://peipa.essex.ac.uk/info/mias.html

[18] "Autoencoder". Wikipedia: The free Encyclopedia. Wikimedia Foundation, Inc. 17th February 2016. Web.

[19] Denoising Autoencoders (DA), (n.d.), http://deeplearning.net/tutorial/dA.html

[20] Vincent, Pascal, et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." The Journal of Machine Learning Research 11 (2010): 3371-3408.

[21] Y. Bengio, Learning Deep Architectures for AI, Foundation and Trends in Machine Learning, vol 2, no 1, pp 1-127,2009.

[22] UFLDL Tutorial: Autoencoders, (n.d.), http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

[23] P. Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), pages 1096 - 1103, ACM, 2008.

[24] “Principal Component Analysis”. Wikipedia: The free Encyclopedia. Wikimedia Foundation, Inc. 16th February 2016. Web.

[25] L.Deng and D.Yu. Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing, vol. 7, nos. 3-4, pp. 197-387, 2013.

[26] Deng, Li. "Three classes of deep learning architectures and their applications: a tutorial survey." APSIPA transactions on signal and information processing (2012).

[27] Palm, Rasmus Berg. "Prediction as a candidate for learning deep hierarchical models of data." Technical University of Denmark (2012).

[28] Qianhaozhe You; Yu-Jin Zhang, "A New Training Principle for Stacked Denoising Autoencoders," in Image and Graphics (ICIG), 2013 Seventh International Conference on, vol., no., pp.384-389, 26-28 July 2013

[29] Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems 19 (2007): 153.

[30] Bengio, Yoshua, Aaron Courville, and Pierre Vincent. "Representation learning: A review and new perspectives." Pattern Analysis and Machine Intelligence, IEEE Transactions on 35.8 (2013): 1798-1828.