UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS PEDRO SILVESTRIN

MR Spectroscopy Signal Quantification Using Deep Learning

Trabalho de graduação.

Trabalho realizado no laboratório Creatis de Lyon dentro do acordo de dupla diplomação UFRGS - INP Grenoble.

Orientador brasileiro:
Profa. Dra. Mariana Recamonde Mendoza

Orientador francês:
Prof. Dr. Michaël Sdika

Co-orientador francês:
Prof. Dr. Helène Ratiney

Porto Alegre
2018

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silvestrin, Luis Pedro

MR Spectroscopy Signal Quantification Using Deep Learning / Luis Pedro Silvestrin. – Porto Alegre: Ciência da Computação da UFRGS, 2018.

69 f.: il.

Trabalho de conclusão (Bacharelado) – Universidade Federal do Rio Grande do Sul. Bacharelado em Ciência da Computação, Porto Alegre, BR–RS, 2018. Orientador brasileiro: Mariana Recamonde Mendoza; Orientador francês: Michaël Sdika; Co-orientador francês: Helène Ratiney.

1. Espectroscopia eletromagnética. 2. Deep Learning. 3. Redes neurais convolucionais. 4. QUEST. I. Recamonde Mendoza, Mariana. II. Título.

RESUMO

Sinais de espectroscopia por ressonância magnética nuclear são utilizados no diagnóstico de doenças importantes tais como Alzheimer, câncer, entre outras. Isso é possível através da quantificação dos metabólitos presentes nos órgãos sujeitos ao exame de espectroscopia. Porém, a presença de ruído e a sobreposição dos sinais emitidos por alguns metabólitos podem tornar o resultado da quantificação impreciso. Neste trabalho, nós implementamos e testamos diversas arquiteturas de redes neurais convolucionais para quantificar sinais de espectroscopia, e comparamos elas com o QUEST, que é a técnica estado-da-arte nessa área, utilizando o erro relativo como medida. Nossos resultados mostram que o erro das redes neurais é aproximadamente 3 vezes menor que o QUEST para sinais contaminados com ruído gaussiano e sinal de fundo. Esse resultado é promissor e mostra que *Deep Learning* é uma abordagem para a quantificação de sinais de espectroscopia que merece ser explorada.

Palavras-chave: Espectroscopia eletromagnética, Deep Learning, Redes neurais convolucionais, QUEST.

Resumo estendido

Este é um resumo estendido em português para a Universidade Federal do Rio Grande do Sul do trabalho original que segue. O trabalho de conclusão original, em inglês, foi apresentado no Institut Polytechnique de Grenoble através do programa de dupla diplomação Brafitec entre as duas universidades.

1 INTRODUÇÃO

Ressonância Magnética (RM) é uma técnica não-invasiva usada para detectar doenças importantes, tais como tumores cerebrais, distúrbios convulsivos do cérebro, ou mal de Alzheimer. Ela funciona através da exposição do órgão examinado a um campo magnético e captura das frequências emitidas pelos diferentes metabólitos presentes no órgão. A RM é subdividida de acordo com o tipo de informação que é usada para extrair: imagem, espectroscopia e imagem por espectroscopia. Neste trabalho, focamos nas informações de espectroscopia.

Diversos metabólitos podem ser encontrados no corpo humano em quantidades diferentes. Um dos principais problemas da espectroscopia por ressonância magnética é quantificar esses metabólitos corretamente para utilizar essa informação em diagnósticos e estudos, pois essas quantidades são um indicador importante de doenças metabólicas. Porém, essa tarefa é dificultada pelas imperfeições no modelo do sinal. Existem componentes adicionais presentes no sinal, como sinal de *background* e ruído que podem levar a resultados incorretos.

Os sinais obtidos da espectroscopia por RM são divididos em duas categorias: sinais com tempo de trânsito longo e sinais com tempo de trânsito curto. O primeiro caracterisa a captura de poucos metabólitos, enquanto o segundo captura sinal de *background* e diversos metabólitos adicionais.

2 QUANTIFICAÇÃO DE SINAIS DE ESPECTROSCOPIA DE RESSONÂNCIA MAGNÉTICA

O sinal de espectroscopia é composto pela soma dos sinais emitidos por cada substância presente no órgão examinado pela ressonância magnética. Dentre essas substâncias, encontramos algumas que são produzidas por reações metabólicas, chamadas metabólitos. Os metabólitos são importantes pois seus níveis de concentração funcionam como indicadores de diversas doenças. Assim, o principal objetivo da quantificação dos sinais de espectroscopia é identificar essas concentrações corretamente.

Além dos metabólitos, a espectroscopia também captura um sinal proveniente de alguma macromolécula presente no local do exame, também chamado de sinal de *background*. Esse sinal não representa nenhuma informação importante e dificulta a identificação dos metabólitos pois possui correlação com os sinais de alguns deles. Somado a isso ainda temos um ruído gaussiano que torna a quantificação ainda mais complicada.

Os sinais provenientes dos metabólitos podem ser modelados utilizando uma equação matemática com parâmetros representando a variação de frequência, atenuação e amplitude dos sinais. Por conhecermos esse modelo, chamamos a parte dos metabólitos de parte paramétrica do sinal de espectroscopia. A abordagem comum para o problema da quantificação é estimar esses parâmetros utilizando otimização por mínimos quadráticos, e é utilizada por diversos algorítmos do estado da arte.

Os sinais são subdivididos em duas categorias: tempo de trânsito longo (LE) e tempo de transito curto (SE). A primeira inclui sinais que capturam poucos metabólitos. O sinal com tempo de trânsito curto, por sua vez, é coposto por diversos metabólitos e também sinal de *background* e, portanto, é mais difícil de quantificar.

Estado da Arte

A aplicação estado da arte para resolver o problema da quantificação de sinais de espectroscopia é chamada QUEST e foi considerado um dos melhores métodos no desafio de quantificação do ISMRM de 2016. Ela se baseia na otimização por mínimos quadráticos para estimar a parte paramétrica do sinal e remove o sinal de *background* através de uma etapa de pré-processamento, na qual alguns pontos do sinal são truncados.

Existem também outras variações do QUEST que utilizam um sinal de *background* adquirido durante a quantificação, sem precisar da truncagem. Outra abordagem simula o sinal de *background* como uma mistura de gaussianas. Chamamos essas abordagens de semi-paramétricas, pois utilizam um modelo paramétrico para o sinal de *background*. Neste trabalho, vamos comparar essas abordagens com o QUEST original e com nossa solução baseada em *Deep Learning*.

3 ABORDAGEM COM DEEP LEARNING AO PROBLEMA DA QUANTIFICAÇÃO

Deep Learning é uma técnica de aprendizado de máquinha supervisionado, ou seja, que necessita da uma base de dados de treino, onde cada exemplo deve estar anotado corretamente. Ela consiste em treinar uma rede neural com diversas camadas através do algorítmo de *backpropagation*. Para o aprendizado funcionar, esse método requer uma quantidade grande de dados que, no caso da espectroscopia, é inviável de se obter. Por isso, nesse trabalho nós simulamos os sinais necessários para treinar e testar os modelos.

Simulação dos Sinais de Espectroscopia

A simulação dos sinais é feita em duas etapas: a simulação da parte dos metabólitos (parte paramétrica) e a simulação do sinal de *background* (parte não-paramétrica). Para simular a parte paramétrica, nós primeiro geramos os parâmetros do modelo aleatóriamente, dentro de um intervalo físicamente realístico, e em seguida aplicamos eles ao modelo. O sinal de *background* é simulado de forma semi-paramétrica, gerando uma mistura de gaussianas que aproxima um sinal de *background* adquirido. Aplicamos então pequenas variações aos parâmetros da mistura para produzir novos sinais de fundo. Por fim, geramos um ruído gaussiano e somamos todas as partes e obtemos o sinal simulado.

Neste trabalho, além de simular o sinal, também utilizamos representações alternativas dele para encontrar aquela que funcione de forma mais eficiente com as redes neurais. As representações utilizadas são: sinal no domínio tempo (t), sinal no domínio frequência (f), ambas as formas concatenadas (TF), o sinal concatenado com subamostras ($subT$), o espectrograma do sinal (g) e, por fim, o sinal com um indicador de sua posição no domínio tempo (*TP*).

Adaptando a Rede Neural à Representação do Sinal

Como cada representação que utilizamos possuem diferenças quanto ao número de dimenções e canais, o modelo de rede neural para cada uma delas precisa de adaptações. Para os sinais no domínio tempo ou no domínio frequência, que possuem apenas um canal de uma dimenção, utilizamos uma rede neural simples com convoluções de uma dimenção. Para representações com mais de um canal, como é o caso de *TF*, *TP* e *subT*, temos que criar uma rede com o número de canais correspondendente com a entrada.

A representação de espectrograma é a única com duas dimenções e, portanto, requer que a rede neural tenha convoluções bidimensionais. As demais representações utilizam apenas convoluções uniimensionais.

4 IMPLEMENTAÇÃO

Para testar as redes neurais utilizamos o *framework Caffe* e para simular os dados utilizamos MATLAB juntamente com o pacote jMRUI, que implementa funções específicas para esse tipo de simulação. A simulação dos sinais de tempo de trânsito longo utiliza apenas 3 metabólitos, equanto 20 metabólitos são usados nas simulações de trânsito curto. Os sinais de base aplicados na simulação são os mesmos disponibilizados no desafio de quantificação do ISMRM de 2016.

A implementação das redes neurais foi feita programaticamente através da API do *Caffe* para a linguagem *Python*. Com ela, é possível criar uma descrição de cada camada e parâmtros em alto nível para que o *Caffe* possa interpretar e executar a rede neural. Além disso, esse *framework* executa em GPU's, o que permitiu treinar de maneira eficiente todas as topologias testadas.

5 EXPERIMENTOS

Para treinar as redes neurais e validar sua performance, simulamos uma base de dados de 200.000 exemplos de treino e outra de 20.000 exemplos de validação, todos esses sinais simulados possuem um nível de ruído variando entre 0%, e 30% da maior amplitude dentre os sinais de base. Para cada um dos 6 tipos de entrada que mostramos no capítulo 3 testamos cerca de 20 topologias diferentes para encontrar aquelas com melhor performance com cada entrada, totalizando 120 topologias. Esse teste foi feito tanto para os sinais com tempo de trânsito longo quanto para os sinais com tempo de trânsito curto.

Para testar as topologias, foram simulados 6 conjuntos de 1000 sinais cada, cada um com um nível de ruído diferente: 30%, 20%, 10%, 5%, 1% e 0%. Com eles, testamos as redes neurais e comparamos seus resultados com o algorítmo QUEST e suas variações. Os conjuntos de teste também foram usados para selecionar as melhores topologias para cada tipo de entrada. A medida usada para a comparação foi o erro quadrático médio entre a amplitude estimada e a utilizada para gerar cada sinal.

No teste usando sinais com tempo de trânsito longo, a comparação mostrou que o QUEST possui uma performance melhor que as redes neurais para quantificar os três metabólitos. Em contrapartida, na comparação com sinais de tempo de trânsito curto as redes neurais tiveram um desempenho até 10 vezes melhor para alguns metabólitos em relação ao QUEST. As redes neurais também se mostraram mais eficientes em tempo de execução: para cada exemplo, o tempo gasto para quantificar com as redes é na ordem dos microssegundos e, com o QUEST, é na ordem dos segundos.

6 DISCUSSÃO

Analisando os resultados, vemos que o QUEST é mais eficiente para quantifcar sinais com tempo de trânsito longo, pois como não há sinal de fundo, a aproximação por mínimos quadráticos tem a vantagem de conhecer o modelo do sinal. Além disso, o QUEST também se mostrou mais tolerante ao ruído nessa categoria de sinal.

Por outro lado, *Deep Learning* mostrou uma performance muito boa para quantificar os sinais com tempo de trânsito curto, bem como um tempo de execução melhor do que o de QUEST. O erro das redes neurais foi bem abaixo do erro apresentado pelo QUEST para sinais com todos os níveis de ruído, e para todos os metabólitos, incluindo aqueles cujos sinais se sobrepõem ou que possuem correlação com o sinal de *background*.

7 CONCLUSÃO

Neste trabalho, foi feita uma comparação de algorítmos de *Deep Learning* para quantificar sinais de espectroscopia de ressonância eletromagnética com um algorítmo estado da arte nessa área, o QUEST. Para isso, experimentamos cerca de 100 topologias diferentes, incluindo combinações de topologias com adaptações do sinal de entrada. Os resultados motraram que as redes neurais são capazes de aproximar as amplitudes dos sinais de cada metabólito com uma precisão maior que o QUEST para sinais com tempo de trânsito curto, onde a quantificação é mais difícil. Com isso, concluímos que o *Deep Learning* tem potencial para se tornar uma ferramenta interessante na quantificação de sinais de espectroscopia.

Para o futuro, novos testes ainda podem ser feitos para validar melhor a performance das redes neurais, como usar um conjunto de sinais de teste gerados com sinais de fundo provenientes de macromoléculas diferentes. Outro teste interessante seria usar o *Deep Learning* para quantificar os demais parâmetros dos sinais (variação de frequência e atenuação) e utilizar o seu output para inicializar o algorítmo de mínimos quadráticos do QUEST.

REFERÊNCIAS

SILVESTRIN, L. P. **MR Spectroscopy Signal Quantification Using Deep Learning**. Disponível em: <http://inf.ufrgs.br/~lpsilvestrin/Silvestrin_LP_master_thesis.pdf>. Acesso em: 29/06/2018.

Master of Science in Informatics at Grenoble
Master Informatique
Specialization Data Science

MR Spectroscopy Signal Quantification Using Deep Learning

Luis Pedro Silvestrin

September 4th, 2017

Research project performed at CREATIS

Under the supervision of:
Michaël Sdika
Hélène Ratiney

Defended before a jury composed of:
Michaël Sdika
Anatoli Juditsky
Bill Triggs

September 2017

Abstract

Magnetic Resonance (MR) spectroscopy signals are used in the diagnosis of important diseases such as Alzheimer's, cancer, and others. This is achieved by quantifying the metabolites present in the signal. However, the presence of background and noise, and the overlap of some of the metabolites, can make the quantification result inaccurate. In this work, we implement and test several convolutional neural network architectures to quantify spectroscopy signals, and we compare them with QUEST, the state-of-the-art approach, using the relative error as measure. Our results show that CNNs achieve an error about 3 times smaller than QUEST for signals containing noise and background. This result is promising and show that Deep Learning is an approach to spectroscopy signal quantification worth to be explored.

Résumé

Les signaux de spectroscopie de Résonance Magnétique Nucléaire sont utilisés dans le diagnostic de plusieurs maladies importantes tels comme l'Alzheimer, cancer, entre autres. Cependant, la présence des signaux de macromolécule et du bruit, et le chevauchement de métabolites peut dégrader la qualité du résultat de la quantification. Dans ce travail, on met en oeuvre et test plusieurs architectures de réseaux neuronaux convolutionnels pour quantifier des signaux de spectroscopie, et on les compare avec QUEST, l'état de l'art pour la quantification, en utilisant la mésure d'erreur relative. Nos résultats montrent que les CNNs atteignent une erreur environ 3 fois inférieure à celui de QUEST pour les signaux qui contiennent bruit et signal de fond. Ce résultat est prometteur et montre que Deep Learning est une nouvelle approche à ce problème qui mérite une étude plus approfondie.

Contents

Abstract **i**

Résumé **i**

1 Introduction **1**
1.1 Magnetic Resonance and MR Spectroscopy 1
1.2 MR Spectroscopy Signal Quantification 1
1.3 Quantification Using Deep Learning 2
1.4 Contents of this Report 3
1.4.1 MR Spectroscopy Signal Quantification 3
1.4.2 Deep Learning Approach to the Quantification Problem 3
1.4.3 Practical Implementation 4
1.4.4 Experiments 4
1.4.5 Discussion of Results 5
1.4.6 Conclusion 5

2 MR Spectroscopy Signal Quantification **7**
2.1 Composition of the Spectroscopy Signal 7
2.2 The Quantification Process 8
2.3 State of the Art 9
2.3.1 QUEST 9
Background in the Basis Set 9
2.4 Related work 9

3 Deep Learning Approach to the Quantification Problem **11**
3.1 Spectroscopy Signal Simulation 11
3.1.1 Background Signal Simulation 12
3.1.2 Alternative Signal Conversions 13
3.2 Deep Learning and Convolutional Neural Networks 13
3.2.1 Fully Connected and Convolutional Layers 14
The Bias Term 15
3.2.2 Max Pooling 15
3.2.3 Inception Module 16
3.2.4 Rectifier Linear Unit (ReLU) 16

PReLU and Leaky ReLU 17
3.2.5 Euclidean Loss Function and Stochastic Gradient Descent 17
Overfitting and Regularization 18
3.2.6 Batch Normalization 18
3.2.7 Dropout 18
3.3 Quantifying Spectroscopy Signals with Deep Learning 19
3.3.1 Adapting the Model to the Input Type 19

4 Practical Implementation **21**
4.1 Data Simulation with MATLAB 21
4.2 Implemention of the Convolutional Neural Networks 22

5 Experiments **23**
5.1 Datasets Used 23
5.1.1 Long Echo-Time Datasets 23
5.1.2 Short Echo-Time Datasets 23
5.2 Relative Error Measure 24
5.3 Correlated Metabolites 24
5.4 CNN Topologies Tested 24
Long Echo-time Topologies 25
Short Echo-time Topologies 26
5.5 Results 28
5.5.1 Results Obtained with QUEST 28
5.5.2 Results Obtained with Deep Learning 29
5.6 Comparing Approaches 30
5.6.1 Long Echo-Time Comparison 31
5.6.2 Short Echo-Time Comparison 31
5.6.3 Time Efficiency Comparison 32

6 Discussion of Results **39**

7 Conclusion **41**

A Appendix **43**

Bibliography **51**

— 1 —
Introduction

1.1 Magnetic Resonance and MR Spectroscopy

Magnetic Resonance (MR) is a non-invasive technique used in the detection of important diseases such as brain tumors, brain seizure disorders, Alzheimer's, among others. It works by exposing the examination subject to a magnetic field, and by capturing the resonance frequencies emitted by the different metabolites in the subject. MR is subdivided according to the type of information it is used to extract: there is imaging (MRI), spectroscopy (MRS) and spectroscopy imaging (MRSI). In this work, we focus on the spectroscopy information.

Different metabolites are present in the human body in varying quantities. One of the main problems of MR spectroscopy is to quantify these metabolites accurately to use this information in medical diagnosis and studies, since their quantities are an important indicator of metabolic diseases. However, this task is complicated by an imperfect modelling of the signal. There are additional components present in the signal, such as background, noise and acquisition artifacts, that can lead to incorrect results.

The signals obtained from MR spectroscopy are divided in two categories: the long echo-time signals and short echo-time signals. The first is composed only by a few metabolites, while the last one includes background and several additional metabolites.

1.2 MR Spectroscopy Signal Quantification

In the quantification problem, the metabolite part is modeled as a parametric function of time, whose parameters represent the extra damping, frequency shift and amplitude of the signal of each metabolite. These parameters vary from subject to subject, and from metabolite to metabolite, and the quantification objective is to estimate them. Finding the three parameters correctly allows us to rebuild the original signal, but only the amplitudes are important to measure the relative amount of a metabolite in the examination subject.

The best existing approaches to this problem, such as QUEST [10], are based on a non-linear least squares fit of the signal model to estimate the three parameters. This technique is shown empirically to be very accurate when applied to long echo-time signals. However, it is not as effective to quantify short echo-time signals. This happens because there isn't a well defined model for the background, so it have to be removed by other techniques before the quantification. Also, the additional metabolites and the background overlap each-other in the signal, making it more difficult to fit correctly with the least squares technique.

The quantification of spectroscopy signals can be seen as a regression problem, where we try to learn a relation between the numerical values of the signal and the amplitudes used to generate the metabolites it contains. For this end, we explore a new approach using Deep Learning and the simulation of the signals.

1.3 Quantification Using Deep Learning

In this work, we experiment with Deep Learning, a supervised machine learning technique, to quantify spectroscopy signals. This technique consists of training a model to return a target value when given the corresponding input example. In our case, the signal is given and the output are the amplitudes of the metabolites it contains. In order to the model to learn the correlation between signal and amplitudes, a large amount of examples and their real amplitudes are necessary. However, in vivo spectroscopy signals are difficult to acquire and their correct amplitudes are unknown, making them unsuitable for the training, so we have to simulate the signals.

The simulation of the parametric part of the spectroscopy data is based on the model used for the non-linear least-squares quantification. It requires to randomly generate the three parameters of each metabolite and scale them according to the real expected proportions. The background, on the other hand, doesn't have any known model, so we first estimate a mixture of gaussians to model an acquired background signal, and then we can randomly vary the Gaussian peaks' parameters to produce different signals. We simulate three independent datasets using this method: one for training of the Deep Learning models, another for validation and a third one for testing.

Deep Learning models are composed of different kinds of layers that succeed each other, passing their outputs forward, from the first input until the last layer's output. These layers represent different operations that are applied to the input, such as convolutions, dot products, dimension reduction, concatenation, normalization, etc. The parameters of all the layers are learned via stochastic gradient descent applied to an objective function that aims to minimize the Euclidean distance between the network output and the target. More specifically, the gradient of this function is calculated for each parameter of the model through an algorithm called Backpropagation.

The layers can be disposed in almost arbitrary topologies and combinations of parameters. To find out the ones that work better, we generate several of these combinations. We also experiment with different input types by converting the signal into its spectrum, spectrogram, and by combining these types, resulting in 6 different input types. In the end, all the combinations of topologies and input types result in over 100 different networks.

Once the topologies are defined, we train them using a training dataset, and, simultaneously, we test them with the validation dataset. This test helps do identify if the network is overfitting with the training data and to find the optimal combination of topology and parameters for each input type. The final test, used to compare the results with QUEST, is done with the test dataset, which is independent from the validation one, in order to avoid networks with hyperparameters overfitting the validation data, thus, keeping a fair comparison.

We evaluate trained networks using the test datasets simulated with different noise levels to assess the noise tolerance of the models. To compare Deep Learning with the state of the art, we use the mean and standard deviation of the relative error over the test datasets. We also use this measure to select the best networks (one of each input type) based on tests with validation datasets.

For the comparison between QUEST and Deep Learning, we divide the tests according to the signal type: long and short echo-time, because of the differences in the composition of both. The long echo-time tests show that QUEST is more efficient to quantify this type of signal, being around 2 times better for signals with noise up to 100 times more accurate when there is no noise. However, Deep Learning show good improvements for short echo-time signals, in the quantification of all the metabolites present. It is able to reduce the relative error of some metabolites by about 3 times, and gives consistent results even when there is noise and overlapping peaks.

1.4 Contents of this Report

This section summarizes the contents of the next chapters of this report, each section being dedicated to one of them. In short, we present the problem, our solution, the experiments and their results. In the end, we also discuss the outcome of the tests.

1.4.1 MR Spectroscopy Signal Quantification

In the chapter 2, we define in details the MR spectroscopy signal quantification problem. The composition of the signal is explained: its mathematical model, its parameters, the basis set that composes it, and the nature of the noise and background that are added to it. The spectroscopy signals as divided in long and short echo-time, according to the presence of background and number of metabolites. Then, in section 2.2, we show how the existing works approach this problem, by using the non-linear least squares fit of the parametric part of the signal.

Next, section 2.3 presents QUEST [10], the state of the art approach and explains its method of background removal by truncating the signal. We also explain other variants that are used to evaluate the Deep Learning approach later in chapter 5. These variants use a semi-parametric model of the background and include it in the non-linear least squares fit. One of them uses the macromolecule signal in the basis set [9], and the other uses the individual peaks estimated from the background signal [8]. In the section 2.4, there is a comparison with another work which uses artificial neural networks for quantification [1].

1.4.2 Deep Learning Approach to the Quantification Problem

Chapter 3 brings more details about how Deep Learning can be applied to quantify spectroscopy signals. In section 3.1, we explain the process of simulating the parametric part and the background of the signal. The parametric part is simulated by generating the parameters randomly and applying them to the model. The background, in turn, requires first to fit a mixture of Gaussians to a macromolecule signal, and then it is used as a parametric model. In the end, random Gaussian noise is scaled according to the noise level and added to the simulation. In section 3.1.2, we explain the 6 input types used in this work: time domain signal, frequency domain, *TF*, *TP*, spectrogram and *subT*.

In the section 3.2, the different types of layers used in the tests are presented. We also explain the choice of parameters, the minimization algorithm (stochastic gradient descent), and the loss function used, which is the Euclidean loss. Summarizing, the layers presented in this chapter are: convolutional, fully connected, max pooling, and batch normalization. In general, we use convolutional layers before the fully connected ones, and we apply batch normalization to all the short echo-time trained networks, and some of the long echo-time topologies. The most used activation is ReLU, but we also experiment with leaky ReLU, PReLU and sigmoid. For regularization, we use dropout in some networks, and $L2$ is also added to the objective function. We also experiment with inception modules, an architecture model from the Deep Learning literature [14].

Section 3.3 explains the topology choices made to build the tests. Finally, in section 3.3.1 we show the adaptations required to use some of the input types with the CNNs, such as splitting the convolutional part of the network and concatenating outputs before passing them to the fully connected layers.

1.4.3 Practical Implementation

This chapter describes the data simulation and how the CNNs were implemented. Section 4.1 explains in details the parameters used in the simulation of the signals. The MATLAB framework was used to this end, since the simulation of the parametric part is implemented in this language, and because of its functionalities for random number generation and matrix manipulation. It is also used for the non-linear least square fit of the Gaussian mixture for the macromolecule estimation.

The section 4.2 introduces the methods used to implement the CNNs, which are the programming language Python and the library Caffe. In the end, we list the parameters used to set up the Caffe solver during the training of the networks.

1.4.4 Experiments

Chapter 5 brings the details about the tests made, the datasets used, topologies tested, as well as the results. The section 5.1 shows how the long and short echo-time datasets are composed, the first are simulated with three metabolites (NAA, Choline and Creatine), while the last ones have 20 metabolites and background added. There are 6 test datasets for both signal categories, each dataset has a different noise level, to assess the noise tolerance of the approaches.

Next, section 5.3 explains the relative error, which we use to measure the accuracy of the quantification methods in the tests, and how it is calculated. To compare the error in the datasets, we use the mean and square root of this measure.

Section 5.4 lists all the topologies used in the tests in this chapter, showing diagrams for each one of them and explaining their details. There is one topology chosen for each one the 6 input types, for long and short echo-time signals, resulting in 12 topologies in total.

The following section shows the results obtained with QUEST on long echo-time signals and with the 5 QUEST variations of background estimation on short echo-time signals. The results appear through plots comparing the mean of the relative error on the 6 noise level test datasets, comparing the error increase with the noise level increase. There are also tables showing the mean and the standard deviation of the relative error for all the short echo-time metabolites. Additional tables and plots with the standard deviation of the error can be found in the appendix A. The same plots and tables were made to compare the CNNs and choose the most accurate ones.

In the section 5.6, the results of QUEST and Deep Learning are compared for long and short echo-time signals. Here we use the same measures and plots of the previous section, but with emphasis in the comparison between both approaches.

In the long echo-time tests, QUEST showed better results for all the metabolites and noise levels, and specially in the tests without noise, where it is about 100 times more accurate than the CNNs. The tests with short echo-time, on the other hand, showed that the CNNs are better at dealing with the background and noise. In these tests, the CNNs using spectrogram and frequency domain data showed the best results, giving results 2 to 3 times better than QUEST for most of the metabolites. They also give consistent results even with addition of noise.

In the last section, we compare both approaches with regard to the time spent during the short echo-time tests. While QUEST takes seconds to quantify one signal, the CNNs take only milliseconds.

1.4.5 Discussion of Results

In the chapter 6 we make observations about the tests of chapter 5, and summarize the results. In short, QUEST shows a much better precision than CNNs for long echo-time signals, while for short echo-time signals the opposite happens. The Deep Learning models are also tolerant to noise and can quantify correctly even overlapping metabolites.

In this chapter we also add some observations about some of the topologies trained. The network trained with time domain data was less noise tolerant than the one trained with *TP*, suggesting that adding the time position to the signal may be an improvement. Also, by comparing the topologies trained with spectrogram data, we see that the short echo-time version, which uses batch normalization, performed relatively better than the long echo-time version, which may suggest that a normalization is required in the latter.

1.4.6 Conclusion

Finally, chapter 7 summarizes the MR spectroscopy quantification problem, the Deep Learning approach and the evaluation method used in this work. It also includes future works suggestions for the Deep Learning Approach, such as using it to estimate the other parameters of the signal, or combining all the parameters in the training. The output of the CNNs could also be used as initialization parameters for QUEST. In the data simulation side, the approach could be extended to generate signals with acquisition artifacts.

— 2 —
MR Spectroscopy Signal Quantification

In this chapter, we explain the MR spectroscopy signal quantification problem, its difficulties, and how the signal is composed. Finally, the state of the art approach, QUEST, is presented along with some of its variants that are also compared to the Deep Learning approach in the next chapters.

2.1 Composition of the Spectroscopy Signal

The spectroscopy signal is obtained through a magnetic resonance exam of a subject (e.g. human brain). The subject of this examination contains certain metabolites that are captured in the signal. In addition to them, the signal is composed of the response of macromolecules, noise and acquisition artifacts.

The metabolites' part is the parametric part of the signal, and its mathematical model is defined in equation 2.1. For each metabolite, there is a basis signal ($x_m(t)$), which is known before the quantification, and the parameters to be estimated: the amplitude (a_m), the damping factor ($\Delta\alpha_m$), the frequency shift (Δf_m), and the overall phase (ϕ_0). The amplitudes give the concentration of the metabolites, so they are the parameter of real interest in the quantification and can be used to study the biochemical modification of patients.

$$\hat{x}(t) = \sum_{m=1}^{M} a_m x_m(t) exp(\Delta\alpha_m t + 2i\pi\Delta f_m t) exp(i\phi_0) \tag{2.1}$$

The non-parametric part, or the background, is a macromolecule present in the subject, whose model is unknown, although, in some cases, it can be estimated. This signal can be acquired during the exam, but at the cost of a prohibitive acquisition time for clinical routine. Usually, the background signal has smooth and smaller peaks than the metabolites (figure 2.1). However, this characteristic can change, depending if the resonance uses a high or low magnetic field. In the end, we can formulate the signal as the equation 2.2, with the background $b(t)$ and the random Gaussian noise e.

$$\hat{y}(t) = \hat{x}(t) + b(t) + e \tag{2.2}$$

We can subdivide the spectroscopy signals in two distinct classes: long echo-time and short echo-time. The first regards the signals that are composed only of a few metabolites, and there is no background, thus being easier to quantify. The short echo-time signals, on the other hand,

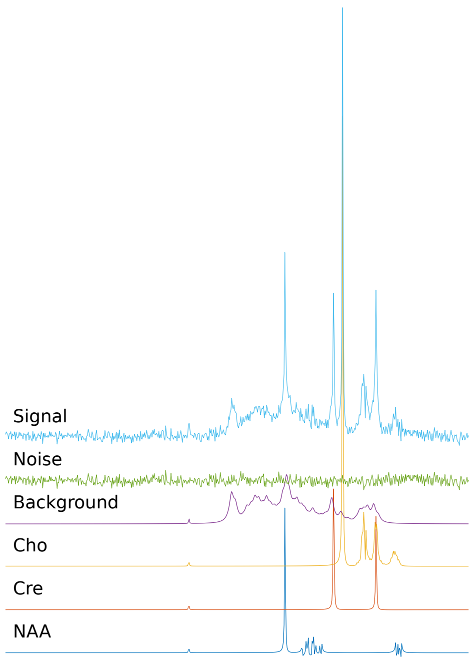

Figure 2.1: Example of signal with three metabolites, noise and background.

contain several metabolites and also have the addition of background, so the quantification of this type of signal is more difficult.

2.2 The Quantification Process

The quantification process consists of finding the parameters (amplitude, damping factor and frequency shift) for each metabolite in a way that the result fits better the input signal. In other words, we want to obtain the values of the parameters given the spectroscopy signal. For this reason, we say that the quantification is a regression problem. If we consider just the parametric part of the signal, the parameters can be easily estimated by doing a non-linear least squares fit (equation 2.3) of the model using the known basis set. However, when the signal contains noise and background, this approach is no longer accurate because this extra information is difficult to model. In addition, the peaks of the background and metabolites can overlap each other (as in the example of figure 2.1), making it harder for the minimization process to find an accurate approximation of the parameters. To deal with this problem, there are different proposed approaches which try to estimate the background.

$$\min ||x(t) - \hat{x}(t)||^2 \tag{2.3}$$

2.3 State of the Art

In this section, we present the current state of the art approaches to spectroscopy signal quantification. We also discuss the similar works that are related to our solution using Deep Learning.

2.3.1 QUEST

A well known approach to the quantification problem is QUEST [10], which ranked between the best methods in the 2016 ISMRM quantification challenge. It uses the assumption that the background signal has a larger damping factor than the metabolites, so it is essentially concentrated in the first points of the signal. Using this prior knowledge, QUEST truncates the first points, producing a residual that contains only the metabolites signal and noise. With this residual, the metabolites can be estimated, and the estimation result is then removed from the original signal. This new residual now contains only background and noise, and from it QUEST estimates the macromolecule signal using singular value decomposition to obtain a parametric version of the background. Finally, this estimated background is removed from the original signal, and the metabolites' parameters can be recalculated with another non-linear least squares fit.

Background in the Basis Set

In addition to the truncation method, QUEST can also be adapted to the case when we have some information about the background. If the macromolecule signal is known, for example, the truncation process can be avoided simply by using this signal in the basis set, allowing QUEST to estimate its parameters the same way it does to the metabolites [9].

Another variant, considering that the background is not known, but is estimated by a parametric model, is to use each peak of the background individually as part of the basis set. For example, the background can be modeled by a Gaussian mixture estimated from a real acquired background signal, where each Gaussian approximates a peak. Also, it is possible to constrain the Gaussian mixture parameters during the quantification process to introduce more physically realistic priors, as is done in [8]. The estimation of the Gaussian mixture for the background will be explained in the section 3.1 as part of the background signal simulation process.

2.4 Related work

There is a previous work which uses neural networks to quantify spectroscopy signals [1]. However, this approach considers only signals with 3 metabolites, and the background have to be removed in a preprocessing step. The only input type used is the spectrum of the signal. In addition, the input of the network should be restricted to the region where the metabolite peak is located, assuming that this is known a priori. This choice disregards the frequency shift parameter from the signal model.

— 3 —

Deep Learning Approach to the Quantification Problem

Deep learning is a supervised machine learning technique broadly applied in classification and regression problems and with successful results in many fields such as computer vision and speech recognition. It consists of training a neural network with arbitrary number of parameters and hidden layers using a loss function through the backpropagation algorithm and using a dataset of labeled examples. In this chapter we explain how deep learning can be used to quantify magnetic resonance spectroscopy signals and all the required procedures to build the network and generate the datasets.

3.1 Spectroscopy Signal Simulation

As any supervised learning technique, deep learning requires a relatively large amount of data to be trained and give reasonable results. However, real MR spectroscopy signals are difficult to acquire, since they require a patient to undergo the examination, and he also have to volunteer the collected data for the experiments. Furthermore, the ground truth parameters of interest are not available with in vivo MRS signals. So, to have enough data to train correctly a deep learning model, we choose to simulate the signals.

The MR spectroscopy signal simulation works differently for the parametric and non-parametric parts. To generate the parametric part, we use the same model from equation 2.1 and choose randomly a different set of parameters for each new signal. The amplitudes are either values from the interval $[0, 1]$, or around 30% of the expected amplitude of each metabolite. The later choice is important when the signal contains background, so that the proportion of each metabolite is realistic compared with the background peaks. The metabolites present in the signal are defined by choosing which basis are used in the model. Finally, with the signals and the parameters, we have the inputs and the ground truth to be compared in the model's objective function.

Since we also want the method to have a certain tolerance to noise, we add random Gaussian values to the signals. To generate data sets with different noise levels, the Gaussian values are scaled by some factor, which have to be proportional to the size of the metabolites in the basis set. We call this factor the noise level (σ), and it is scaled by the largest value in the basis set and the largest value among all the expected amplitudes to have the correct proportion.

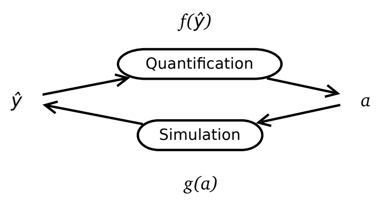

Figure 3.1: Simulation as the reverse of the quantification process, with a being the set of parameters of the signal.

3.1.1 Background Signal Simulation

Differently from the parametric part, in a real spectroscopy signal, the background is composed of a macromolecule signal of unknown model. To simulate it in the most accurate way, we approximate this signal with a Gaussian mixture model, so that we can have a separate Gaussian curve for each peak of the signal (figure 3.2). With this approximation, it is possible to vary the parameters of each Gaussian separately and produce distinct background signals.

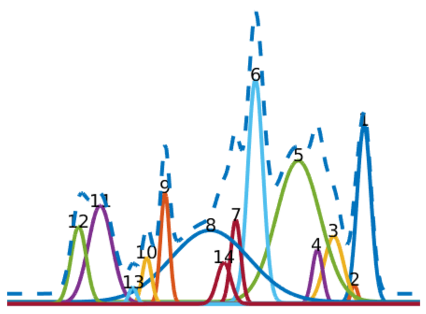

Figure 3.2: Example of the peaks produced by the simulation, with the original background as the dashed line.

$$b(t) = \sum_{m=1}^{M} a_m exp\left(-(\alpha_m t)^2 + 2i\pi f_m t\right) \tag{3.1}$$

The model used to approximate the background signal is the equation 3.1. It represents M Gaussians, with a_m, α_m and f_m being, respectively, the amplitude, damping factor and frequency shift of the signal, and t representing the position in time. In this model, the damping factor is squared to make the signal decrease faster than the metabolites, resulting in a signal that can be estimated by QUEST via the truncation method.

Finally, after simulating the background, noise and metabolites' signals, all this parts are summed to form a new signal. In the next sections, we explain how to set up a Deep Learning model to work with the simulations.

3.1.2 Alternative Signal Conversions

The versatility of Deep Learning models allows us to use almost any input format for training and testing. In addition to the standard time domain signal, there are experiments with different adaptations of the signal in the frequency domain (or spectrum). We also use the spectrogram of the signal, which is a 2-dimensional vector composed of Fourier Transforms of several overlapping portions of the signal. Each row represents the frequency domain while the columns represent a portion of the original signal subject to the transform. Figure 3.3 shows an example of spectrogram, with the signal peaks concentrated in the bottom-left corner.

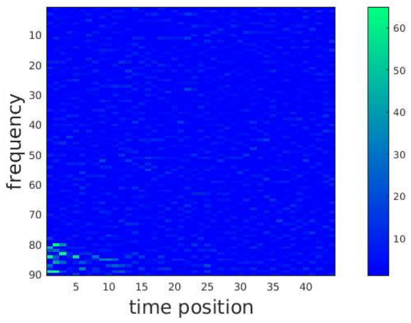

Figure 3.3: Spectrogram of a short echo-time spectroscopy signal.

Other variations tested with the time domain signal include subsamples (*subT*), time and frequency (*TF*) domains together and time domain with a time position indicator (*TP*). The *subT* input is produced by subsampling the original signal twice, one with half the number of points and another with one fourth (e.g. a signal with 2048 points will have 1024 and 512 points as subsamples), both are inputted together with the original signal. The *TP* input has an additional channel that indicates explicitly the position in time of the points, as is shown by figure 3.4. The *TF* input is the concatenation of the time domain and frequency domain

Signal	s(1)	s(2)	···	s(N)
Time Position	1	2	···	N

Figure 3.4: Diagram exemplifying the *TP* input. $s(n)$ refers to the time-domain signal at time position n.

versions of the same signal as two separate channels of the input. In the next sections, we explain how to adapt the Deep Learning model to work with these inputs.

3.2 Deep Learning and Convolutional Neural Networks

Deep learning algorithms are based on artificial neural networks, but they can include an arbitrary number of hidden layers, each one followed by a non-linear activation function. In the class of Deep Learning algorithms, there are the Convolutional Neural Networks (CNNs), which includes the use of convolutional layers. The basic building blocks of convolutional neural networks are the fully connected (and/or convolutional) layers, activation functions and a loss function required for training. There are also the max pooling layers, which help to reduce the size of the input and introduce non-linearity to the learned model. Other components commonly found in CNNs are the dropout, that helps to regularize the model during the training, and the batch normalization layers, used to normalize the outputs and improve the model's efficiency.

Neural networks are trained through the backpropagation algorithm in two phases, the forward step and the backward step. During the forward step, examples are given to the network, and it calculates the output of each layer, from the first to the last one, to finally obtain the loss value. Then, during the backward step, this value is used to compute the gradient of the loss function with respect to each layer parameter, from the last layer to the first, using the chain rule. The next sections explain how each layer of the model is picked and how to set up the parameters.

3.2.1 Fully Connected and Convolutional Layers

A fully connected layer (or FC layer) works like a hidden layer from a multi-layer perceptron. It is composed of M sets of weights (for an output of size M), called neurons, and each set has the same size as the input vector, so it is possible to do a dot product between them. The input of this kind of layer is always in form of vector, and, because of the nature of the dot product, the spatial information in the input is lost in the output, this means that, if the input information is shifted during test, it will can give a bad result. In CNNs, these layers learn a linear transformation of the outputs of the convolutions to obtain the probability of the input to belong to each class, in the case of classification, or the estimation of a value, in the case of regression.

Convolutional layers can receive either matrices or vectors as input, and they can also have multiple channels. For example, we can have an input with dimensions $MxNxD$, where MxN is the matrix dimension and D is the number of channels. The weights of this layer are the kernels used during the convolution, and they can have an arbitrary shape MxN, but must have the same number of channels as the input, so each kernel multiply and sum all the channels of the input and map them to a new channel in the output. The convolution also has a stride parameter S, meaning that each multiplication of the kernel is spaced by S positions from the previous one. In order to control the size of the output, there is a padding parameter P, which inserts an extra contour of zeros around the input. The output of this layer have as many channels as the number of kernels, and its shape is of size in each dimension is $(W-K+2P)/S+1$, where W is the size of the input and K is the size of the convolution kernels.

Different from the fully connected layer, the convolutional layer keeps the spatial information of its input in the output. This means that a kernel that learns to respond to a particular pattern in the input will always give the same output, and it will have a position relative to the input. For example, if the input is a spectroscopy signal with a background peak shifted, a convolutional layer will detect this peak and give an output with a similar shift, so the spatial information is propagated to the next layers. For this reason, it is more interesting to keep this kind of layer before the fully connected layers.

The Bias Term

Both fully connected and convolutional layers include a bias term which is learned like every other parameter. This term is summed to the final result of the layer without being multiplied by the input, so, in the trained model, it represents a constant shift in the output values (red term in equation 3.2).

$$z = \sum(w_i y_i) + w_0 \tag{3.2}$$

However, during the quantification of the amplitudes, this shift is undesired because, when the input signal is scaled compared to the training set, the outputs of the network should also be scaled. In this case, the bias term will shift the output, resulting in incorrect quantification. For this reason, this term is removed in every layer of the networks trained for the quantification of the amplitudes.

The shifting behaviour can be observed in a test made with two neural networks with same number of layers and neurons, but one of them using the bias term. Both were trained with the same dataset of time domain examples, and the test in the figure 3.5 was made using another dataset with the amplitudes scaled by 5, using 1000 examples. The figure illustrates how the histogram of errors has its center moved when the bias term is used, reflecting a shift in the correct output.

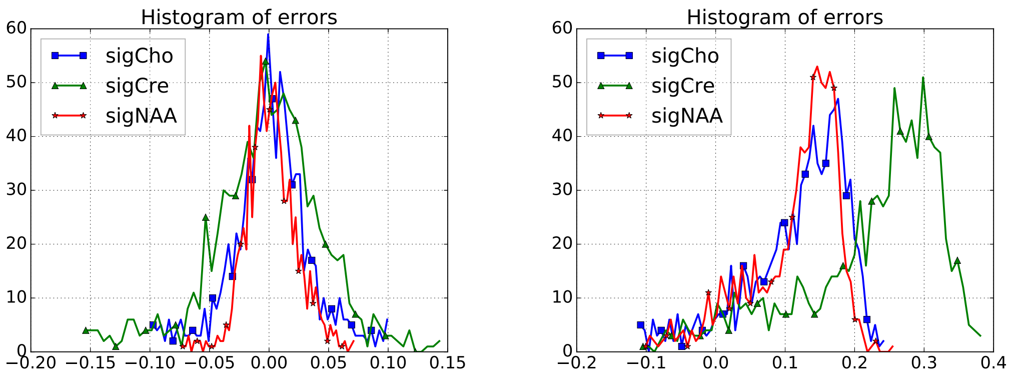

Figure 3.5: Two histograms comparing one network without bias (left) and another with bias (right). Each color represents the amplitude of a different metabolite. The y-axis tells the number of examples and the x-axis, the relative error.

3.2.2 Max Pooling

In convolutional neural networks, the input has to pass through several layers, until the final layer can process an output, such as a class probability or, in the case of quantification, the proportion of a metabolite. Generally, there is a large difference between the dimension size of the input and the output, and the layers have to gradually reduce the size of the input until it reaches the output. To have a good result, it is recommended that this dimension reduction process be as smooth as possible [14]. Smaller inputs are also better for the efficiency of the model, since it will take less time to convolve a smaller input. In such cases, it is useful to insert max pooling layers in the network. This kind of layer works in a way similar to the convolutional layers, having a kernel size and stride parameters, but instead of multiplying, it returns only the largest value of the input region under the kernel. Other main difference is that it pools values channel-wise, so its output always keeps the number of channels of the input.

3.2.3 Inception Module

Convolutional neural networks are the current state of the art technique for image classification problems, and this field is constantly producing new advances to improve the topologies. One of them is the inception model, developed in the GoogLeNet [13] architecture.

On a recent work [14], the inception technique is redesigned to increase the efficiency and the quality of the neural networks. Its main idea is to smoothly reduce the size of the layer outputs and increase the number of dimensions, making the network deeper and losing less information in the middle of the computations. In its examples, the work suggests to process the input in three parallel subnetworks, as is shown by figure 3.6. In practice, it uses convolutions with 1×1 kernels to reduce the input dimensions by half without reducing the input size (and preserving part of this information). Afterwards, it uses a convolution with stride 2 to reduce the the input size by half, preserving the number of channels. This is done in two branches of the module, and the third one does a max-pool to the input, and the three outputs are concatenated channel-wise to produce a new output with the double of channels and half the size of the input.

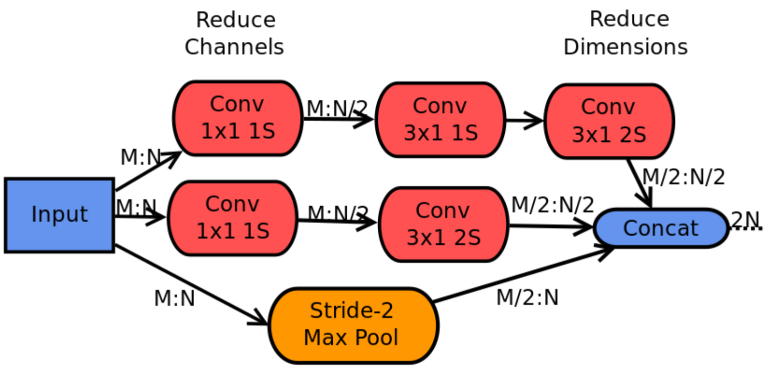

Figure 3.6: Inception module example for 1-dimensional inputs. M is the input size and N is the number of dimensions

3.2.4 Rectifier Linear Unit (ReLU)

The activation function is an operation applied point-wise to the output of the fully connected or convolutional layers. It is used to restrict the output of the previous layer to a range of values, defining which outputs will be passed on to the next layer. There are several activation functions such as *tanh*, or sigmoid, that restrict the output to $(-1,1)$ and $(0,1)$, respectively, but this kind of activation can let the output saturate close to its limits, resulting in very small gradient value and, therefore, making training slower.

To avoid this behaviour, a common practice in deep neural networks is to use he rectifier linear unit (ReLU) [2], defined as in equation 3.3. Unlike sigmoid and hyperbolic tangent functions, this activation has no upper bound, so it can forward larger values without saturating. On the other hand, it can "die" when the neuron output is zero, and in this case the gradient would also be zero, so then neuron can not change.

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{3.3}$$

The ReLU is the activation used in most of the tests in this work because of its success in many previous deep learning works, and due to its relation with proofs of the universal approximation property of neural networks [11]. Nevertheless, we still use sigmoid in some of the topologies implemented, and it also gives interesting results, as is shown in the chapter 5.

PReLU and Leaky ReLU

There are other variations of the ReLU that aim to solve the problem of the dying gradient when it reaches the zero value, such as the PReLU [3] and Leaky ReLU. The leaky ReLU modifies the negative part of the original ReLU function by returning the same input value scaled by a small constant (e.g. 0.1). This way, no matter how small the input is, it will not be set to zero and the gradient will be passed back through the activation. The PReLU, on the other hand, is a generalization of the leaky ReLU which allows the model to learn a parameter instead of using a constant for the negative part, is shown by equation 3.4.

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases} \tag{3.4}$$

3.2.5 Euclidean Loss Function and Stochastic Gradient Descent

The loss function chosen in this work is the Euclidean loss (or mean squared error), defined as in equation 3.5. In the equation, N represents the number of examples, $\hat{y}$ is the prediction given by the network and y is the ground truth we try to approximate.

$$E = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{y}_n - y_n||_2^2 \tag{3.5}$$

The Euclidean loss is appropriate to train neural networks to regression tasks, as is the case of the quantification of spectroscopy signals. Therefore, we use this loss function to train the neural networks in this work.

Once the loss function is defined, the training occurs through a gradient descent algorithm. On each step of this algorithm, a batch of examples is inputted in the network, their resulting loss is calculated and the gradient is computed for every weight of the model using the back-propagation algorithm. The gradient is multiplied by a learning rate, and then subtracted from its respective parameter. This process, called Stochastic Gradient Descent (SGD), goes on until all the parameters converge or until a maximum number of iterations is reached.

During the execution of SGD algorithm, the minimization of the objective function usually oscillates in different directions before it reaches a minimum, making the process to require more iterations to finish. To accelerate the learning, a common technique is to add to the gradient some portion of the gradient from the last iteration (equation 3.6).

$$\begin{aligned} v_t &= \beta v_{t-1} + \gamma \nabla_\theta E(x) \\ \theta &= \theta - v_t \end{aligned} \tag{3.6}$$

This extra value is called momentum, and it was already shown how effectively it increases the speed of the SGD algorithm [7].

At some point during the descent, it is a common issue that the gradient gets stuck in a couple of values, unable to continue decreasing until the minimum, or that the loss function start to increase. This may happen when the gradient value is increasing and the network is being updated with bigger values. A straight forward solution to this problem is to divide the learning rate by some value as the training advances, so that the updates slowly decrease, and the minimization can continue without getting stuck. In the section 4.2 we give the details about the parameters used during the training of the implemented CNNs.

Overfitting and Regularization

A recurrent problem in many learning algorithms is when the model fits so well the training data that its generalization error starts to increase. This problem is called overfitting, and for Deep Neural networks, and there are special changes to the topology and to the solver that help to cope with it. One of them is to add a regularization term to be minimized with the loss function. In this work, we use the $L2$ regularization, which penalizes the model by summing the $L2$ norm of the model parameters to the objective function (equation 3.7).

$$E = \frac{1}{2N}\sum_{n=1}^{N} \|\hat{y}_n - y_n\|_2^2 + \gamma\|W\|_2^2 \tag{3.7}$$

The parameter γ is tuned to increase or decrease the influence of the regularization term to the training

3.2.6 Batch Normalization

In neural networks with several layers and unbounded activation functions, it can happen that the outputs reach very large values, resulting in errors during the calculation. Other problem is the variation of the distribution of the inputs of each layer, which can slow down the learning process, and requires the tuning of other parameters of the solver. Batch normalization [4] is a layer created to approach these issues by normalizing the output of each layer. It is also shown that batch normalization helps to reduce the model overfitting [4].

During the training phase, the batch normalization layer takes the mean and the variance of current input batch and then normalize its examples, so the output batch has zero mean and unit variance. Then, the output examples are scaled by a parameters learned during the training, so the model is able to regulate how much of the normalization will be passed to the output. After the training phase, when the network is being tested, the layer uses an estimation of the mean and variance of the training set batches. This is done to keep the outputs of the network deterministic.

3.2.7 Dropout

The Dropout layer [12] is another approach to the overfitting problem. It is used during training, and it "turns off" randomly a given amount of parameters of a layer, so that only part of them will participate in the forward and backward phases of the training. The removed outputs are changed during each iteration, so the layer has different parameters updated by the gradient descent. This way, in every iteration, we are training a slightly different model, and the number of these models increase exponentially with the number of parameters we have. In theory, the chance that each of these models learns the same parameters is very low, resulting in a low chance of the final model to overfit.

3.3 Quantifying Spectroscopy Signals with Deep Learning

It has been proven that Deep Learning is capable of approaching arbitrary functions, given enough data and the right network design [11], and, based on this idea, we want it to quantify spectroscopy signals. To do this, the first step is to produce a training and a validation data sets via simulation, as was shown in the previous sections, and, for each example, we keep its metabolites' amplitudes as the labels to be used in the training. Differently from the non-linear least squares fit approach, which have to approximate all the parameters of the signal, the neural network can be trained to approximate only the amplitudes, which are the parameters of interest.

Once we have a training dataset, we choose the topology of the network, the amount of convolutional and fully connected layers and their configuration. All the convolutional layers are kept before the fully connected ones to preserve spatial information through the convolutions. The activation functions are placed after every layer, except by the last FC layer, which should give the quantification result as output.

The number of channels of each convolutional layer increases gradually from the first to the last layer, and the number of outputs of the fully connected layers decrease in the same direction. We also choose the number of neurons of the first FC layer proportional to the number of channels of the output of the last convolution, while the last FC layer always has as many neurons as the amount of metabolites to quantify.

In between the convolutions and before the activations, max pool layers can be eventually placed to reduce the size of the outputs and make the training faster. Batch normalizations are inserted right after convolutional or FC layers, or before the first convolution, to normalize the input directly. Oftentimes they bring considerable improvements when placed after and/or before the first FC layer, since it usually has the largest input of the network, and so its output is also large. This is required specially in the case of short echo-time signals, when several metabolites are summed together with the background, and the amplitude values are larger.

3.3.1 Adapting the Model to the Input Type

In section 3.1.2, different adaptations of the spectroscopy signal were presented. The original time domain signal, as well as its frequency domain version can be inputted directly into the first convolution of the model without any changes. The *TP* type, which is just the time domain signal with an additional channel, also does not require modifications of the neural network. All these inputs are composed of 1-dimensional vectors, so they are convolved with kernels of a single dimension, the only exception is the spectrogram of the signal. In this case, it is necessary to change the kernel of the convolutions to a 2-dimensional format, but the network topology does not need to be changed. *TF* and *subT*, on the other hand, need special alterations in the model.

For the *TF* input, the convolutional layers of the network have to be split in two parallel subnetworks, one for the time domain and other for the frequency domain. The outputs of both subnetworks are concatenated together and used as input for the fully connected layers, as is shown by figure 3.7.

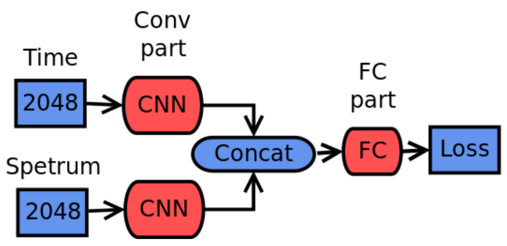

Figure 3.7: Convolutional network split into time and spectrum part.

To process the *subT* input, we used three parallel subnetworks, one for the whole signal, and two others for the subsamples. The first and the second are merged when the outputs of both match the same feature map size, and this new output is then concatenated with the result of the third subnet (figure 3.8). After the last concatenation, the final result is passed to the FC layers. It is also possible to merge the three outputs in a single concatenation before the last FC layer.

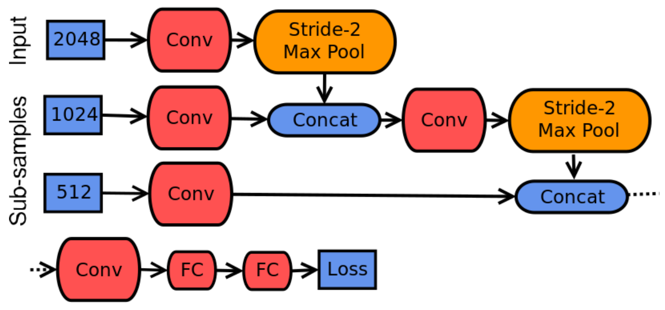

Figure 3.8: The *subT* input is processed in three separate subnetworks.

— 4 —
Practical Implementation

To put the Deep Learning approach in practice and compare it with QUEST, we implement several convolutional neural network topologies, and also simulate the training, validation and testing datasets. We use the MATLAB® (2016a, The MathWorks, Natick, MA) framework to simulate the data, and the Caffe library [6] to train and test the network topologies with the different input types. In the next sections, we discuss how these technologies are put in practice to this end.

4.1 Data Simulation with MATLAB

Section 3.1 shows how the parametric part and the background of spectroscopy signals can be simulated. In addition, the signal can be converted to other representations that are used as other input types to the CNNs. To simplify these tasks, we use the MATLAB framework, which contains all the mathematical tools required to run the models previously seen. Existing MATLAB code was used to simulate the parametric part, while the background simulation was implemented, also in the same language.

The long echo-time signals were simulated using a normalized basis set, where each signal was divided by its maximum value. On the other hand, the short echo-time signals were used without any normalization to keep the proportions of the metabolites and background closer to the realistic values. This is important, so QUEST can produce reasonable results by quatifying these signals.

The basis set signals are all stored into a file format that can be read from MATLAB. These files were taken from the 15th MR spectroscopy quantification challenge, from the ISMRM workshop of 2016 [5]. The model of the metabolite signal is given to the software straightforwardly, then the random parameters are generated to produce the parametric part. The ranges of the parameters of long echo-time signals are $[-10, 10]Hz$ for the frequency shift, $[-10, 10]Hz$ for the extra damping and amplitudes ranging between $[0, 1]$. For short echo-time simulation, we use the extra damping between $[0, 20]Hz$ to avoid a positive damping, since this basis set have a smaller damping than the long echo-time basis set. To keep a realistic proportion between the metabolites and the background, generate the metabolites' amplitudes between $\pm 30\%$ of the values in table A.1 for short echo-time.

To produce the background peaks, we use a macromolecule signal stored in a file by the same way as the basis set. With this signal and the model shown in the previous chapter, MATLAB can make a non-linear least squares fit to learn the parameters of the peaks, which are stored. Later, during the background simulation, the new signals are generated by multiplying these parameters by randomly generated values. The ranges of the simulated background parameters, relative to the original peaks, are: $\pm 30\%$ for the amplitudes, $\pm 5Hz$ for the frequency shift and $\pm 10Hz$ for the damping factor.

The random Gaussian noise is drawn from a normal distribution and then scaled by a parameter σ, as was explained in section 3.1. For the training and validation datasets, σ varies between $[0, 0.3]$ for each example. In the test datasets, this parameter is fixed to generate the same noise level for all the examples.

Once the time domain signal is ready, we use the built-in functions of MATLAB to calculate its spectrum, spectrogram and subsamples. *TF* data have time a and spectrum data stored as separate channels. The same is done for the subsamples of *subT* data. The real and imaginary parts of the signal are kept in independent channels as well. All this information is stored using the HDF5 file format [15], which facilitates the data access by Caffe.

4.2 Implemention of the Convolutional Neural Networks

The framework chosen to implement the convolutional neural networks is Caffe [6]. It is an open source library implemented in C++ aiming efficiency and modularity. The software supports training and testing on multiple GPUs, exploring the parallelism of vector multiplications and convolutions, which are frequent operations in CNNs. It also implements all the required layers and features cited in chapter 3, such as dropout, batch normalization, ReLU and the Euclidean loss function. In addition, it has a Python API which allows to programmatically generate new network topologies and choose the training and validation datasets. This can be used, for example, to test the same network topology with different datasets automatically, allowing to find out which topology works better with which input type. Caffe also gives several useful informations about the training, such as the value of the loss function for the training and validation datasets, and the learning rate at every iteration, which can be used to draw conclusions about the network overfitting.

To prepare a CNN model to be used by Caffe, we simply have to write the layers, their parameters, the loss function and the location of the train and validation datasets in a configuration file. The layers can be specified in any order that does not conflict with their input and output sizes, and without producing cycles in the topology.

Once the model is ready, then it is necessary to prepare a configuration file with the parameters of the solver. In this work, we set it to use the SGD algorithm for optimization, with starting learning rate of 0.01, momentum of 0.9, and 5×10^{-4} for the L2 regularization term. The learning rate is divided by 10 every 20.000 iterations, and the algorithm runs for 100.000 iterations. The size of the batch used for each iteration is 100 examples.

— 5 —
Experiments

To evaluate the Deep Learning approach and compare it with QUEST, we prepared tests with different kinds of datasets. In all tests, we assess how accurately the approach can quantify the metabolites' amplitudes by using the relative error metric. We also verify how tolerant it is to noise, by using datasets with different noise levels. In the rest of this chapter, we discuss the metrics and datasets used in the tests. Finally, we present the results obtained with each approach.

5.1 Datasets Used

In order to the CNNs to work properly, the training datasets have to contain a relatively large amount of examples. We find the size of 200.000 training examples gives reasonable results, and using more than this did not bring greater improvements. For validation, we use 10% of the training set size, or 20.000 examples. Both training and validation sets contain examples with varying noise levels: the noise parameter σ ranges from 0 to 0.3, to ensure that the learned model is tolerant to noise.

There are 6 test datasets, each one with 1000 examples and a different noise level (σ): 0, 0.01, 0.05, 0.1, 0.2 and 0.3. All the datasets are simulated with the same parameters, the only difference from one to the other is the noise scale. These datasets are used to compare Deep Learning and QUEST approaches in both accuracy and noise tolerance.

5.1.1 Long Echo-Time Datasets

The datasets of long echo-time signals are simulated using three metabolites in the basis set: NAA, Choline and Creatine. These basis are normalized before simulation to have the largest value equal to one. In practice, they are composed of 1024 complex-valued data points.

The amplitudes values are limited between 0 and 1. Since the basis are normalized and the maximum amplitude is 1, then the noise for this type of signal is scaled by the exact value of σ.

5.1.2 Short Echo-Time Datasets

Short echo-time datasets are composed of 20 metabolites (see table A.1) and also have a background signal added. The basis signals are not normalized, and their maximum value from all the metabolites is around 10. This value, and the maximum value among the expected amplitudes, are used to scale the noise level σ, so it is proportional to the scale of the signal. The basis are signals of 2048 complex data points, as well as the simulated examples.

5.2 Relative Error Measure

The relative error is the metric used to calculate the accuracy of the models. It is described in equation 5.1, with a being the estimated amplitude and $\hat{a}$ being the ground truth value.

$$E_n = \frac{a_n - \hat{a_n}}{\hat{a_n}} \tag{5.1}$$

Since the difference is normalized by the ground truth, it gives a value that can be compared between the amplitudes of all metabolites, even if they have different expected values, as is the case of short echo-time signals.

With the relative error it is possible to measure the precision of the prediction in one example. To extend this to the whole dataset, we calculate the mean and the standard deviation (equation 5.2) of the relative error over all the examples.

$$\begin{aligned} M_E &= \frac{1}{N}\sum_{n=1}^{N}|E_n| \\ S_E &= \sqrt{\frac{1}{N}\sum_{n=1}^{N}|E_n - M_E|^2} \end{aligned} \tag{5.2}$$

Finally, using these measures, we have a standard way of comparing the accuracy of the tested CNNs in the validation set, and the best CNNs with QUEST in the test datasets.

5.3 Correlated Metabolites

In the case of short echo-time signals, there are pairs of metabolites whose peaks happen to overlap each other, making it more difficult to quantify their amplitudes. When this happens, the quantification usually is more accurate for the peak formed by the sum of both metabolites than the their individual peaks. For this reason, we also calculate the error using the sum of the amplitudes of the correlated metabolites. This way, it is possible to measure the accuracy of the estimation of the peak formed by the overlap.

5.4 CNN Topologies Tested

In this section we explain in details the different CNN topologies tested, for each input type. We subdivide them according to the type of signal used during training: long and short echo-time, because of the difference in the number of metabolites and presence of background. Also, not all the networks that worked well in the first case were as effective in the second case, and vice-versa. For both cases, there are tests with all the 6 input types. Several different networks were tested for each input type and the most accurate ones were picked to compare with QUEST.

The best topologies are chosen based on the mean relative error they have over the validation dataset to avoid the chance of using a network that overfits the test data.

Long Echo-time Topologies

We tested around 20 different network topologies with time domain, frequency domain, *TF* and *TP* input types and 5 topologies with spectrogram and *subT*. Each test varies from the other, in topology aspects, such as number and order of layers, number of parameters, or by using additional layers such as batch normalization, dropout, etc. The networks picked for each input type are referred as the following: *LE-t* (time domain), *LE-f* (frequency domain), *LE-TP* (*TP*), *LE-TF* (time and spectrum), *LE-g* (spectrogram), and *LE-subT* (*subT*).

For both time and frequency domain input types, the best topology is the same, the only difference is the activation function: PReLU for spectrum data and ReLU for the original signal. A particularity of this topology is the absence of activation after the last convolution layer. The disposition of the layers is shown by figure 5.1.

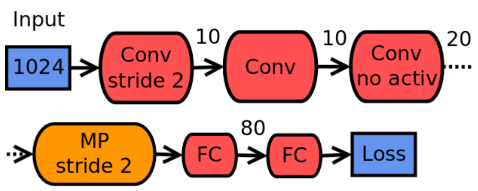

Figure 5.1: Best network topology used for time domain and frequency domain inputs. For time domain it uses ReLU as activation, while for frequency domain it uses PReLU.

The topology of *LE-TF* is illustrated in figure 5.2. The convolutional part (from the first convolution until the max-pool layer) is repeated twice, one for the time signal and other for the spectrum, as is described in chapter 3. The outputs from both parts are concatenated and this result is passed to the two following fully connected layers.

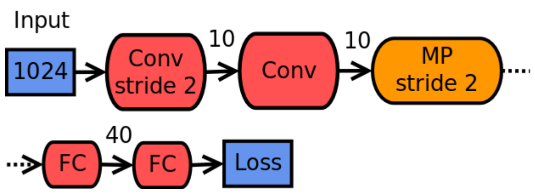

Figure 5.2: Topology trained with *TF* input type. The convolutional part is above, and the fully connected, below.

The *LE-TP* topology contains 11 convolutions, 4 max-pooling layers and two fully connected layers in the end (figure 5.3. Instead of traditional ReLU activations, this topology uses leaky ReLU activations, with 0.01 as parameter for the negative part. In addition, it has batch normalization layers after every convolution and fully connected layers, except by the last one. There is also a batch normalization layer before the first convolution.

The spectrogram topology *LE-g* uses three inception modules (as described in section 3.2.3), and has a total of 18 convolutions (figure 5.4). All the convoltions are 2-dimensional with 3×3 kernels, except by the first convolution, which has a 1-dimensional kernel and stride 2 to reduce the size of the largest input dimension (64). Padding is used in some inception layers to correct the size of the outputs before concatenation. Since this topology has more neurons than the others, we also dropout in the FC layers with 50% chance of dropping each parameter.

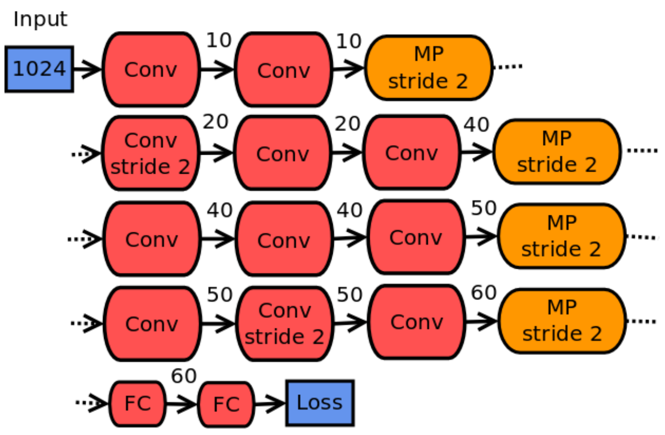

Figure 5.3: Most performant topology trained with *TP* input type.

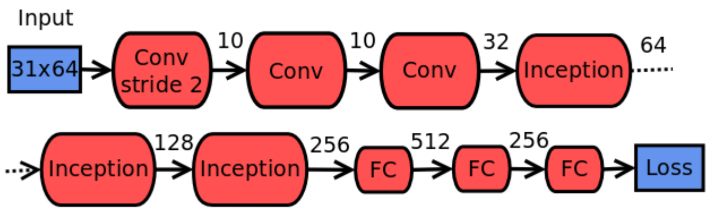

Figure 5.4: Best topology for the spectrogram of the signal.

The best topology for the *subT* input contains 6 convolutions split in three branches, one for the input and two for the subsamples. It differs from the other topologies tested for the same input type, where branches are merged between the convolutions (see section 3.3.1), this network reduces all the inputs inputs to vectors of same size with a fully connected layer before concatenating them.

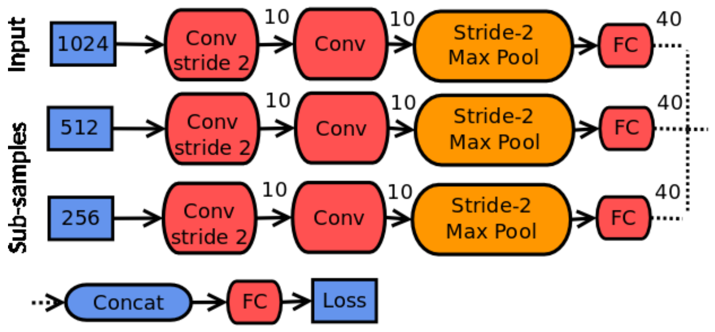

Figure 5.5: Network trained with *subT* input type.

Short Echo-time Topologies

The short echo-time signal contains 20 metabolites and also have a background signal, which increase the difficulty of the quantification. In addition, the basis used to simulate these examples are not normalized, so signals resulted from the simulation contain large numerical values.

Thus, to avoid an exaggerated increase of the output values, we use batch normalization in most of the layers of the topologies tested. We label the networks tested as: *SE-t* (time domain), *SE-f* (frequency domain), *SE-TP* (*TP*), *SE-TF* (time and frequency domain), *SE-g* (spectrogram), and *SE-subT* (*subT*).

The topology of *SE-t* has the same setting of convolutional and fully connected layers as is shown by figure 5.2, but it does not have the split convolutions as *LE-TF*, and has 80 neurons in the second-to-last fully connected layer. This network has sigmoids as activations and batch normalizations after every convolutional and FC layer, except the last one, and after the input. Here, the batch normalization helps the sigmoid function by forcing the outputs to the range $[-1,1]$ and avoiding the saturation of the activation.

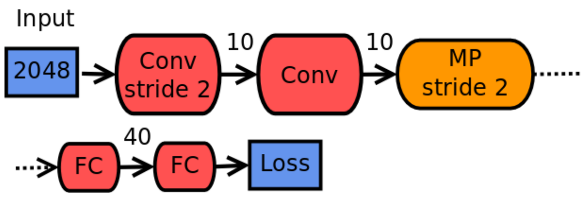

Figure 5.6: *SE-t* topology.

SE-f topology uses 20 convolution layers, 15 of them distributed in 3 inception modules. It uses a batch normalization layer after the first FC layer, adaptation required to short echo-time signals to avoid overflowing values during the training. The model with the output sizes is in figure 5.7.

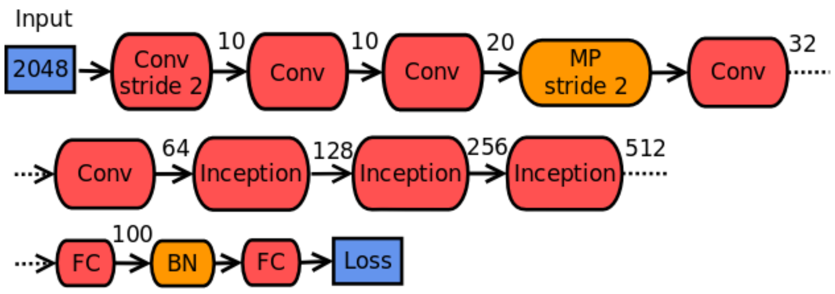

Figure 5.7: *SE-f* topology.

The network of *SE-TF* is composed of 22 convolutions and 8 max-pooling layers split in frequency and time domain subnetworks. Both have the layers distributed as in figure 5.8. It uses batch normalization in all layers and the input.

The *SE-TP* network has the same architecture as *LE-t* (figure 5.1). The only difference is the addition of a batch normalization layer after the second-to-last fully connected layer.

SE-g and *LE-g* share the same topology with the same inception modules (figure 5.4). *SE-g* requires a short echo-time adaptation: a batch normalization after the penultimate FC layer.

SE-subT and *LE-subT* also have the same ordering of layers, as is shown by figure 5.5. As all the other short echo-time networks, *SE-subT* is altered by including a batch normalization after the FC layers of each branch.

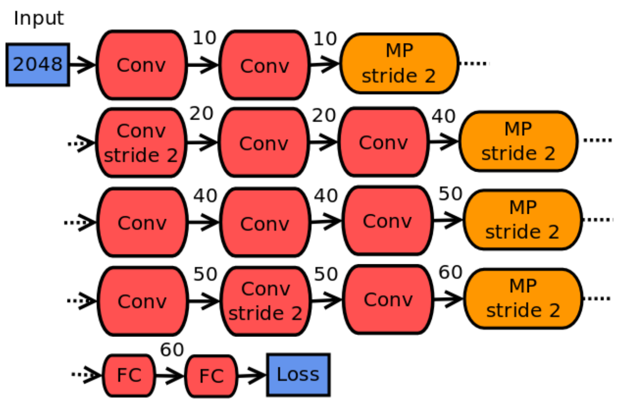

Figure 5.8: *SE-TF* topology.

5.5 Results

In this section, we list the results obtained using QUEST and the networks previously cited. We first show the isolated results of each approach, and in the end we compare both.

5.5.1 Results Obtained with QUEST

We tested QUEST with long and short echo-time signals and calculated the relative error between the amplitudes it quantified and the ground truth used in the simulation. For short-echo time, there are tests with the 5 variations of background estimation, while there is a single test for long echo-time because the background is not present. All approaches are tested with the 6 noise level test sets.

The long echo-time test result is shown by figure 5.9 with a plot for each metabolite of the signal. In this test, we use only a simple version of QUEST, without truncation. singular value decomposition or additional basis signals for background estimation. The plots show Choline in red, Creatine in green and NAA in blue. By this plot we can see that Choline is more affected by the raise of the noise level.

The short echo-time test regards 20 metabolites and 5 different variations of QUEST, echo one with a different approach to the estimation of the background. We refer to the QUEST variations as: *trunc*, *basis*, *peaks*, *const* and *const t25*. *trunc* truncates the first 25 points of the signal to remove the background before the quantification, as is explained in section 2.3.1. *basis* uses the background signal in the basis set during quantification (first part of section 2.3.1). *peaks* uses the background Gaussian peaks in the basis set, while *const* also uses constraints in the parameters of the Gaussians (see section 2.3.1). Finally, *const t25* is based on *const* and also uses the truncation of the first 25 points, like *trunc*.

The tables 5.1 A.2 show the mean and standard deviation obtained in the tests with 0 noise and 0.3 noise, respectively. Among all the metabolites of the signals, there are 6 pairs which are strongly correlated. Thus, we also calculate the error regarding the sum of the amplitudes of these pairs.

By comparing the error obtained for each metabolite in the table with their respective expected amplitudes (table A.1 in the appendix), we can verify that the error is larger for the components with lower amplitude values. These metabolites are usually more difficult to quantify and sometimes can be occluded by the noise.

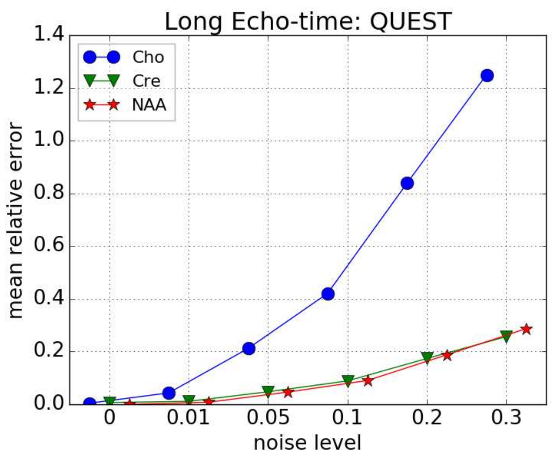

Figure 5.9: Mean relative error of QUEST running with the long echo-time datasets. *Cho* stands for Choline and *Cre* for Creatine

From the values presented by table 5.1, we can see that *const* and *peaks* give more accurate quantifications for most of the metabolites, compared to the other tested techniques. *trunc*, which is the only QUEST variation that does not use the background as a parametric part of the signal, is behind in almost every test. Nonetheless, it is still close the the other approaches regarding the metabolites with larger amplitudes.

The effect of noise in QUEST results is shown by the plots in figure 5.10, where all the 5 variations are tested with 4 different metabolites. The metabolites chosen are Ins, NAA, Cr and GPC, the first three having large amplitudes and are, thus, more reliable for quantification, while GPC has a slightly smaller expected amplitude. For all the 4 metabolites, *basis* and *const* have the best accuracy, while *peaks* and *const t25* are slightly less accurate. *trunc* has similar accuracy to them when quantifying Ins and GPC, but is relatively inaccurate for the other two metabolites.

The *const* variation performed very well on all the datasets, except the one with 0.01 noise level. In this test, QUEST failed during the quantification of 442 out of 1000 examples. The mean relative error shown by the plots for this case takes into account only the signals quantified correctly, without failures. Of all the QUEST variations and noise level tests, this is the only case where such result was observed.

5.5.2 Results Obtained with Deep Learning

In this section we expose the results obtained using the Deep Learning models previously cited with the same test datasets used to evaluate QUEST. Again, we start with the long echo-time results and then we add the background and extra metabolites to the tests.

By the plots in figure 5.11 we see that the topology trained with spectrogram data is the least effective in all the 3 metabolite tests. After it, *LE-subT* present higher error rates for Creatine and NAA, and Choline when the noise level is low. Time, spectrum, *TP* and *TF* present similar error rates in most of the tests, with *LE-t* being the worse in the NAA tests. In most cases, *LE-f* and *LE-TP* have the best results.

To compare the short echo-time results, as there are too many metabolites, we draw plots with the noise level and mean relative error for NAA, Ins, Cr and GPC (figure 5.12), as was done

Metabolite	trunc	basis	peaksMM	constr	constr t25
Ala	1.89±1.47	1.51±1.35	0.62±0.43	**0.61**±0.46	0.70±0.48
Asc	1.79±1.70	0.63±0.50	0.58±0.59	**0.55**±0.51	0.74±0.77
Asp	2.50±1.87	0.98±0.78	1.50±1.59	**0.90**±0.93	1.14±1.21
Cr	0.58±0.52	**0.36**±0.35	0.47±0.45	0.38±0.41	0.41±0.42
GABA	2.34±2.44	1.41±1.50	**0.84**±0.87	1.01±1.22	1.22±1.47
GPC	0.49±0.34	**0.47**±0.32	0.52±0.32	0.47±0.32	0.48±0.32
GSH	1.30±0.94	0.55±0.39	0.55±0.40	**0.50**±0.39	0.59±0.45
Glc	2.69±2.63	2.03±1.96	**1.19**±1.26	1.35±1.55	1.53±1.70
Gln	6.72±5.82	2.84±3.32	**1.59**±2.15	1.79±2.49	2.02±2.81
Glu	0.54±0.35	**0.27**±0.23	0.51±0.33	0.32±0.26	0.38±0.29
Gly	0.94±0.91	0.74±0.78	0.65±0.70	**0.61**±0.69	0.68±0.72
Ins	0.29±0.24	**0.25**±0.22	0.31±0.31	0.27±0.29	0.31±0.29
Lac	18.45±17.32	13.01±15.11	7.58±10.51	**5.57**±8.26	6.58±9.37
NAA	0.25±0.20	0.14±0.13	0.13±0.20	**0.11**±0.16	0.14±0.19
NAAG	3.17±2.47	1.10±1.12	0.96±1.83	**0.83**±1.35	1.04±1.77
PCho	0.97±0.86	**0.86**±0.76	0.86±0.76	0.97±0.83	1.04±0.98
PCr	0.43±0.38	**0.28**±0.28	0.38±0.36	0.29±0.31	0.30±0.30
PE	2.69±2.23	**1.40**±1.35	1.68±2.01	1.70±1.84	1.76±1.82
Tau	0.74±0.50	0.61±0.41	0.66±0.47	**0.59**±0.42	0.99±0.93
sIns	**0.70**±0.56	0.79±0.81	0.73±0.68	0.71±0.62	0.86±0.78
Cr+PCr	0.30±0.19	**0.11**±8.76e-02	0.23±0.23	0.13±0.14	0.15±0.14
GPC+PCho	0.34±0.26	0.30±0.23	0.38±0.27	**0.30**±0.23	0.34±0.27
Glc+Tau	0.74±0.69	0.51±0.47	0.53±0.47	**0.50**±0.48	0.90±0.88
Glu+Gln	0.88±0.40	**0.21**±0.17	0.39±0.27	0.22±0.19	0.32±0.25
Ins+Gly	0.23±0.18	**0.15**±0.12	0.22±0.21	0.18±0.17	0.20±0.19
NAA+NAAG	0.38±0.20	0.12±9.62e-02	0.10±0.11	**9.52e-02**±9.06e-02	0.11±0.10

Table 5.1: Table with the mean and variance of the relative error obtained from the short echo-time tests with 0 noise dataset. The metabolites are represented in the lines and the QUEST variations are in the columns. The last 6 lines contain the error the the correlated metabolite pairs.

before with the QUEST results. From the plots, it is pointed out that *SE-TF* and *SE-TP* have the poorest result when the noise level increases. The network trained with time domain signals also has a relativlely higher mean error for examples with noise. Based on these metabolites, we can define *SE-f* and *SE-g* as the most stable approaches in all noise conditions, and the most accurate in general. This is confirmed for all metabolites in tables A.5 and A.3.

5.6 Comparing Approaches

In this section we put QUEST and Deep Learning side by side using the approaches that presented the best results in the tests comparison of the previous sections. Again, we start with the long echo-time signals and then advance to the short echo-time datasets. In the last section, we also draw a comparison between the execution time of both approaches for short echo-time

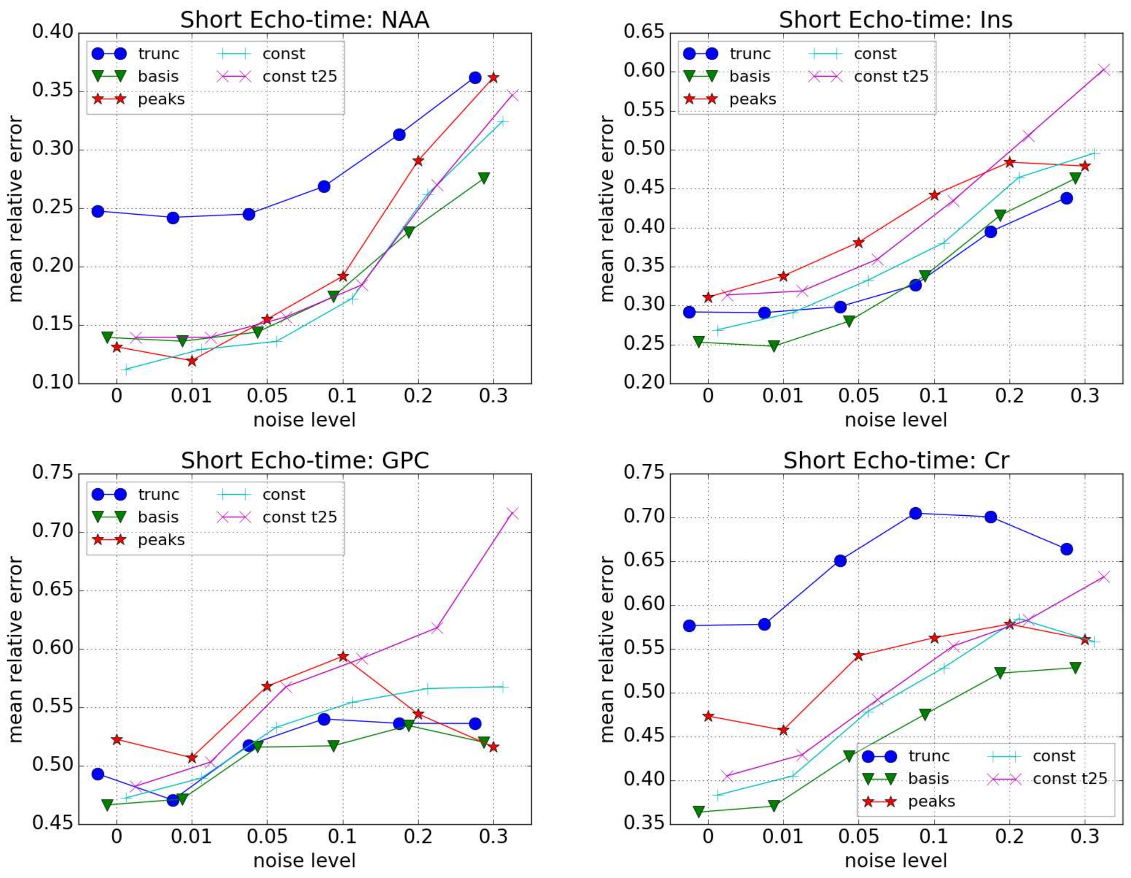

Figure 5.10: Plots comparing the raise of the error on NAA, Ins, GPC and Cr when the noise increases for all short echo-time QUEST variations.

quantification.

5.6.1 Long Echo-Time Comparison

From the Deep Learning models, we pick *LE-f* and *LE-TP*, since they had the best results for long echo-time data, to compare with QUEST. In figures 5.13 and A.1, QUEST has a better accuracy in quantifying all the three metabolites, even in the presence of noise. Without noise, the results are far more accurate than the CNNs. For the NAA metabolite, QUEST achieves an error about 100 times smaller than the CNNs. This can be explained by the fact that the long echo-time signals have few metabolites, there aren't any overlapping metabolites, and there is no background, so the non-linear least squares fit becomes much simpler than it is in the case of short echo-time signals.

5.6.2 Short Echo-Time Comparison

For the short echo-time comparison, we select *SE-t*, *SE-f* and *SE-g* from the CNN models and *basis* and *const* from QUEST variations. We use the mean of the relative error (figure 5.14) as well as the standard deviation (figure A.2) to plot the accuracy of these approaches.

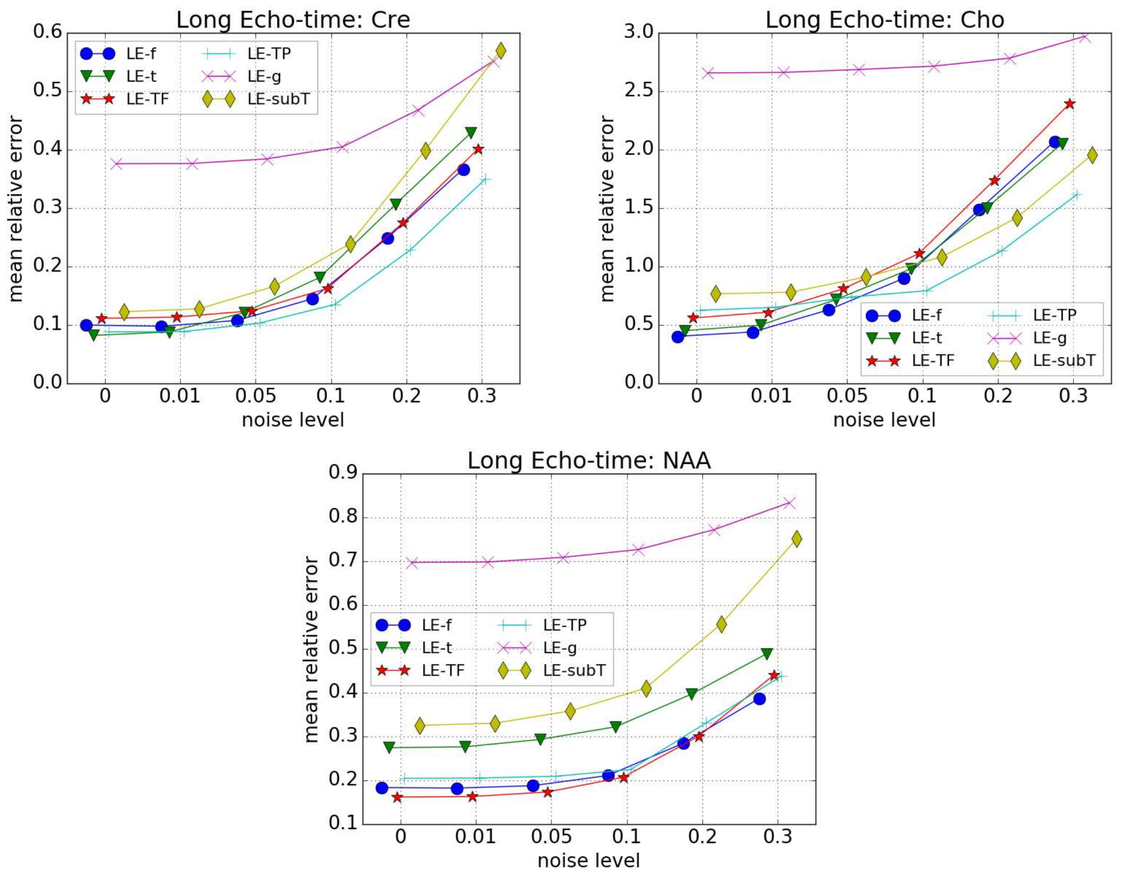

Figure 5.11: Mean relative error of the CNNs tested with the long echo-time test datasets.

Differently from what is observed in the long echo-time comparison, the neural networks improve significantly the quantification accuracy of most of the metabolites. Also, the results of *SE-f* and *SE-g* are very tolerant to the addition of noise, showing only a small increase between 0 (table 5.2) and 0.3 (table A.4) noise datasets, for most of the metabolites. *SE-t* also shows improvements for several metabolites, mainly when there is no noise, but its error starts to increase already at the 0.01 noise test, as is shown by 5.14.

The statistics in table 5.2 show that the CNNs can quantify correctly even the metabolites with overlapping peaks, such as NAA and NAAG. In addition, they give better results to metabolites which are difficult for QUEST to quantify, such as Glc, Gln and Lac.

5.6.3 Time Efficiency Comparison

In addition to the quantification accuracy, we also compared the execution time of both approaches during the tests. All QUEST variations were executed in a Intel Xeon E3-1246 v3 CPU with 3.50GHz of clock, using a single core, while the CNNs were trained with a NVIDIA Tesla K80 GPU and tested with the same CPU.

The times recorded from the tests of the QUEST variations and the CNNs with all the 6 test sets are listed in table 5.3. While the CNNs took seconds to run through all the 6000 examples, the QUEST variations had to run for several hours to finish running.

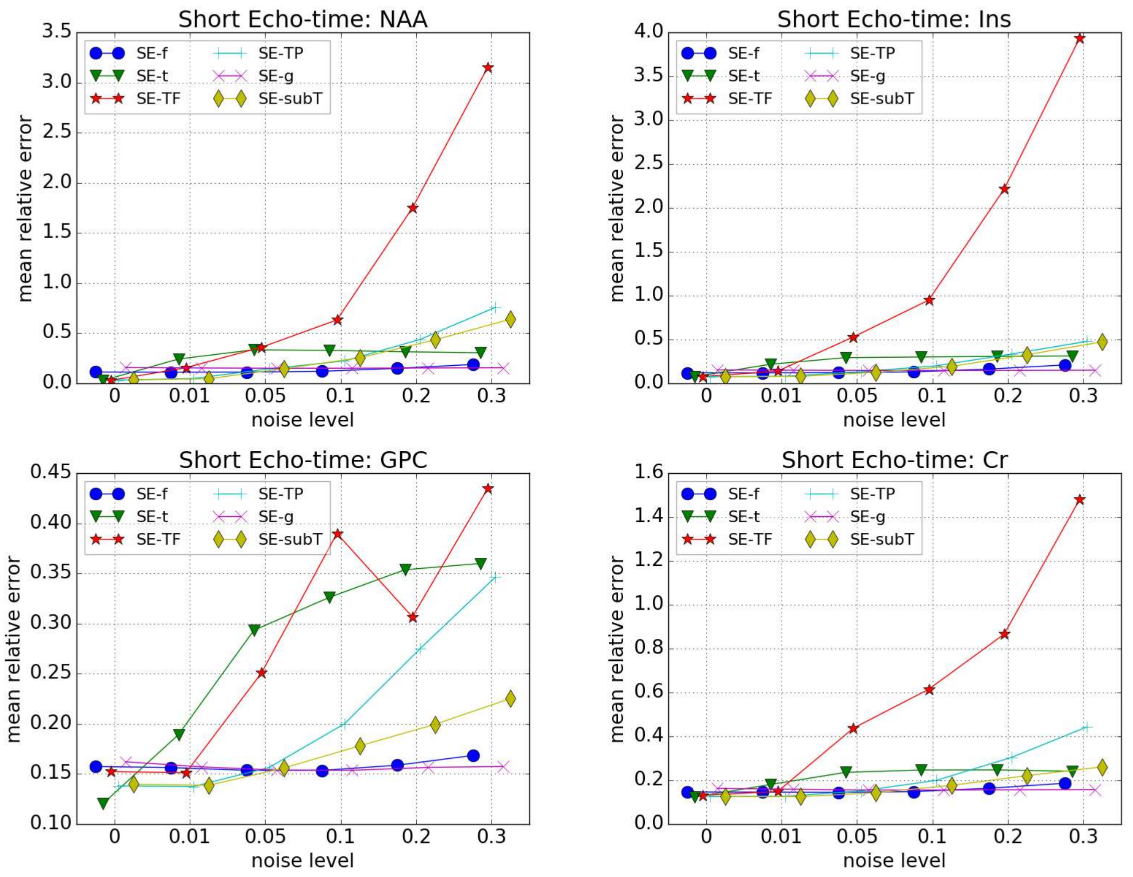

Figure 5.12: Plots comparing the raise of the error on NAA, Ins, GPC and Cr when the noise increases for all short echo-time CNNs.

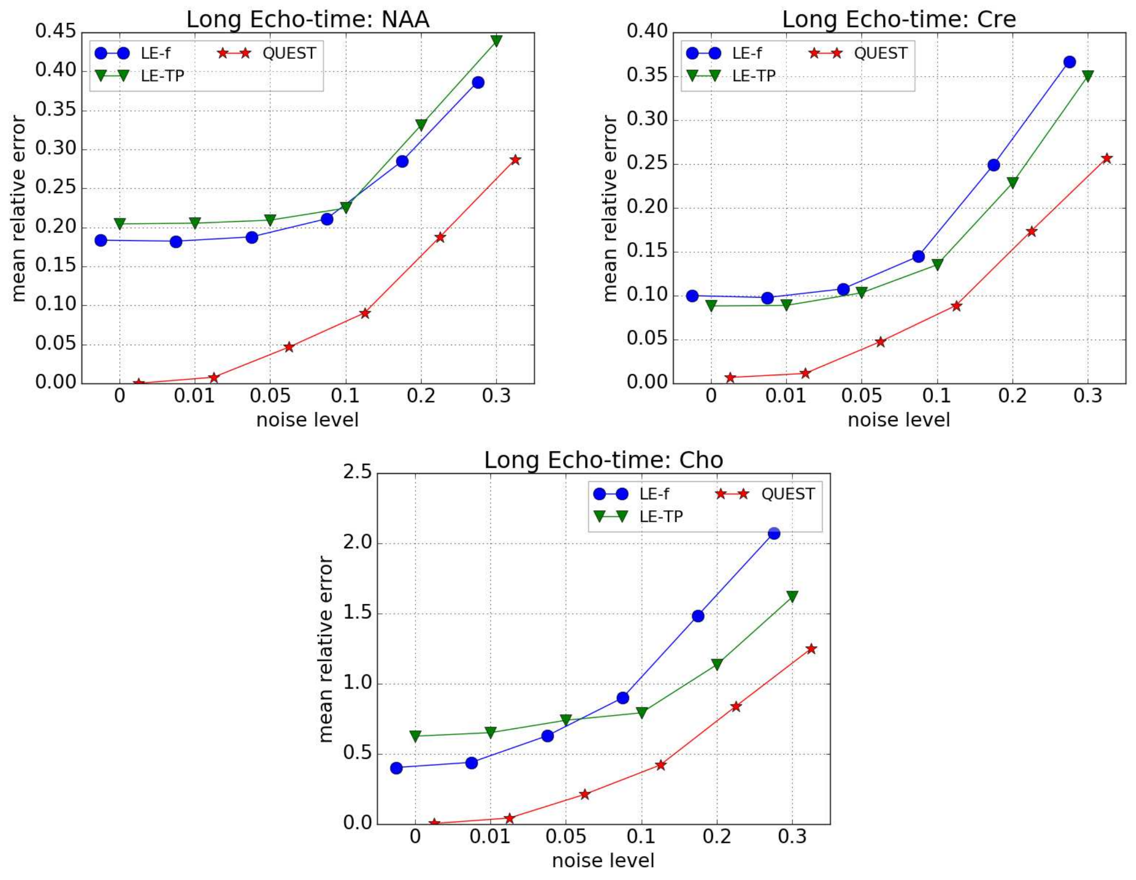

Figure 5.13: Plots comparing the mean relative error of QUEST and the CNNs *LE-f* and *LE-TP* on the test datasets.

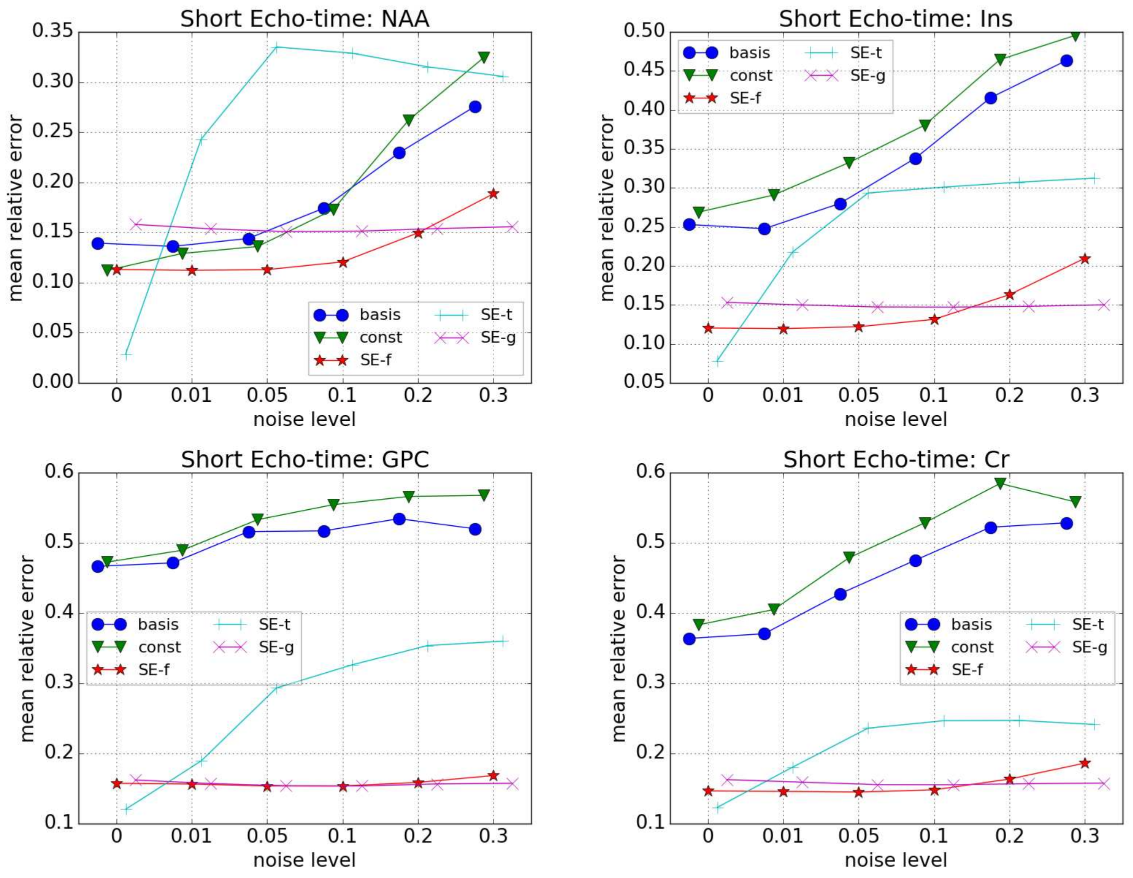

Figure 5.14: Plots comparing the accuracy of QUEST approaches *basis* and *const* with the CNNs *SE-f*, *SE-t* and*SE-g* using the metabolites NAA, Ins, GPC and Cr.

Metabolite	basis	constr	SE-t	SE-f	SE-g
Ala	1.51±1.35	0.61±0.46	**0.15**±0.10	0.16±0.12	0.16±0.11
Asc	0.63±0.50	0.55±0.51	**0.16**±0.10	0.17±0.12	0.17±0.12
Asp	0.98±0.78	0.90±0.93	**0.15**±0.10	0.16±0.11	0.16±0.11
Cr	0.36±0.35	0.38±0.41	**0.12**±8.86e-02	0.15±0.10	0.16±0.11
GABA	1.41±1.50	1.01±1.22	**0.16**±9.77e-02	0.16±0.11	0.16±0.11
GPC	0.47±0.32	0.47±0.32	**0.12**±8.62e-02	0.16±0.11	0.16±0.12
GSH	0.55±0.39	0.50±0.39	**0.15**±0.11	0.16±0.11	0.16±0.11
Glc	2.03±1.96	1.35±1.55	**0.16**±0.10	0.16±0.12	0.16±0.11
Gln	2.84±3.32	1.79±2.49	**0.15**±0.10	0.16±0.11	0.16±0.11
Glu	0.27±0.23	0.32±0.26	**4.87e-02**±3.94e-02	0.12±9.04e-02	0.15±0.11
Gly	0.74±0.78	0.61±0.69	**0.15**±0.10	0.16±0.12	0.16±0.11
Ins	0.25±0.22	0.27±0.29	**7.83e-02**±6.38e-02	0.12±9.20e-02	0.15±0.10
Lac	13.01±15.11	5.57±8.26	**0.16**±0.10	0.16±0.11	0.16±0.11
NAA	0.14±0.13	0.11±0.16	**2.87e-02**±2.17e-02	0.11±8.80e-02	0.16±0.11
NAAG	1.10±1.12	0.83±1.35	**0.15**±9.97e-02	0.15±0.11	0.15±0.11
PCho	0.86±0.76	0.97±0.83	**0.15**±0.10	0.16±0.11	0.16±0.11
PCr	0.28±0.28	0.29±0.31	**0.10**±8.33e-02	0.13±9.23e-02	0.16±0.11
PE	1.40±1.35	1.70±1.84	**0.15**±0.10	0.16±0.12	0.16±0.11
Tau	0.61±0.41	0.59±0.42	**0.14**±0.10	0.17±0.12	0.17±0.11
sIns	0.79±0.81	0.71±0.62	**0.15**±0.10	0.16±0.12	0.16±0.11
Cr+PCr	0.11±8.76e-02	0.13±0.14	**5.35e-02**±4.06e-02	8.15e-02±6.20e-02	0.11±8.58e-02
GPC+PCho	0.30±0.23	0.30±0.23	**8.69e-02**±6.38e-02	0.12±8.70e-02	0.12±9.13e-02
Glc+Tau	0.51±0.47	0.50±0.48	**0.11**±7.76e-02	0.13±9.12e-02	0.13±8.99e-02
Glu+Gln	0.21±0.17	0.22±0.19	**4.27e-02**±3.38e-02	0.10±8.08e-02	0.14±9.82e-02
Ins+Gly	0.15±0.12	0.18±0.17	**6.17e-02**±4.86e-02	9.56e-02±7.41e-02	0.12±8.49e-02
NAA+NAAG	0.12±9.62e-02	9.52e-02±9.06e-02	**2.65e-02**±2.05e-02	0.10±8.04e-02	0.15±0.10

Table 5.2: Table comparing QUEST and Deep Learning approaches using 0 noise dataset.

CNN	Train (hh:mm)	CPU Test (s)	Per Example (ms)
SE-f	6:05	60.09	10.02
ST-t	1:04	20.08	3.35
SE-TF	8:20	54.31	9.05
SE-TP	1:26	20.69	3.45
SE-g	7:38	80.04	13.34
SE-subT	3:37	19.32	3.22

QUEST	Test Time (hh:mm)	Per Example (s)
basis	11:09	6.69
trunc	24:09	14.49
peaks	33:15	19.95
const	49:12	29.52
const t25	51:11	30.71

Table 5.3: In the left, the train and test time of all the short echo-time CNNs, and, in the right, the test time of all QUEST variations. The CNN test time considers the tests made on CPU. The "Per Example" columns give an estimation of the time to quantify one single example.

The time difference shown by the tables is expected, since the non-linear least-square fit of several parameters is time consuming, as well as the singular value decomposition. On the other hand, convolutions and dot products are relatively fast operations. Also, the use of optimized libraries and parallel calculation give an extra advantage to the convolutional neural networks.

Training the neural networks is what consumes more time. The time required to train each CNN, also shown by table 5.3, is proportional to the number of layers of the network. The test times in table 5.3 where measured by testing the CNNs with the same datasets and CPU used by the QUEST variations. The CNNs that took more time (*SE-TF*, *SE-g* and *SE-f* during training are also the ones with more convolutional layers. However, the training is required only once to have a working CNN that can quantify an arbitrary number of examples very fast.

— 6 —
Discussion of Results

The results from chapter 5 show that QUEST is more efficient to quantify long echo-time signals, when there is no overlap of the metabolites and no background. The non-linear least-squares fit takes full advantage of the known parametric model, and can estimate all the parameters much more accurately than the CNNs. QUEST is around 2 times more accurate than Deep Learning for signals with noise, and can be about 100 times better when the signals without noise.

On the other hand, Deep Learning performs the quantification of short echo-time signals better than QUEST, and in much less time. It has a smaller relative error for all the metabolites, and for some of them it can be about 3 times more accurate than QUEST. This shows that the trained neural networks are capable of handling the non-parametric parts of the signal very well. In addition, they also have excellent results even when there is noise and overlapping metabolites.

However, not all the short echo-time networks tested worked as expected: the ones trained with *TF* and time domain inputs showed bad quantification results, mainly with the noisy data. This may suggest that the noise occludes more the input in the time domain than in the frequency domain. The *TP* input, on the contrary, does not have the same issue, proposing that the addition of the time position channel is useful to localize the parts of the signal important for the quantification.

Among the long echo-time examples tested, the *LE-g* topology was less accurate than all the other tests. The same network, when tested with a batch normalization with short echo-time data, performed very well compared to the others. This may suggest that the outputs of *LE-g* require a normalization.

— 7 —
Conclusion

MR spectroscopy signal quantification consists of estimating the amplitudes of the metabolites present in the result of the examination. The correct quantification is important for medical diagnosis of diseases such as cancer and tumors. QUEST, the state of the art approach to this problem, is very precise, however, the presence of background, noise, and overlap of the signal peaks make it difficult to quantify certain metabolites.

In this work, we present a new approach to the spectroscopy signal quantification: training deep neural networks to estimate the metabolites' amplitudes. To do this, we simulated training and validation datasets using the existing signal model, and we also extend this simulation to generate background signals. The networks are implemented using Caffe, a very efficient Deep Learning library, and we create over 20 different topologies, combined with 6 variations of the signal as input, resulting in over 100 trained networks. The most effective networks are selected for each input type and then compared.

We also test quest with long and short echo-time simulated signals with different noise levels. For the short echo-time signals, we test and compare 5 variations of QUEST, each one with a different method to treat the background. The most accurate ones are picked to compare with the deep neural networks.

Finally, we gather the results obtained with all the tests and put QUEST and Deep Learning side by side. By analyzing the accuracy of both approaches, we find out that long echo-time signals are better quantified by the non-linear least squares fit of QUEST. On the other hand, the deep neural networks excel in quantification of short echo-time signals, and is also noise tolerant.

With these results, we expect that Deep Learning will be explored more thoroughly for spectroscopy quantification by finding better topologies and parameters. Also, Deep Learning could be tested to estimate the extra damping factor and frequency shift of the metabolites, and combine this with the amplitude quantification to obtain better results. Other option is to use the outputs of the neural networks as initialization for QUEST's non-linear least squares fit. Regarding the signal simulation, one could also extend it to include the acquisition artifacts, which are another non-parametric part of the signal that could be learned by machine learning models.

— A —
Appendix

Metabolites	Proportion
Ala	1.1
Asc	1
Asp	1.6
Cr	4.15
GABA	1.5
Glc	0.6
Gln	1.28
Glu	13.08
GPC	0.9
GSH	1.76
Gly	2.2
Ins	7.1
Lac	0.09
NAA	14.23
NAAG	1.24
PCho	0.37
PCr	5.55
PE	1.47
sIns	0.41
Tau	1.77
Macromolecule	11.7

Table A.1: Proportion (expected amplitudes) of each metabolite and the background in a short echo-time signal.

Metabolite	trunc	basis	peaksMM	constr	constr t25
Ala	12.36±6.02	3.05±3.13	2.64±2.95	2.52±2.58	**2.23**±2.72
Asc	2.35±2.26	**2.30**±1.97	2.82±2.53	2.45±2.14	3.08±2.98
Asp	4.67±3.65	**2.94**±2.52	6.07±4.46	3.42±2.83	4.60±5.68
Cr	0.66±0.62	**0.53**±0.46	0.56±0.40	0.56±0.42	0.63±0.56
GABA	11.64±7.06	3.61±3.63	**2.14**±2.30	2.71±2.79	3.25±4.47
GPC	0.54±0.47	0.52±0.36	**0.52**±0.36	0.57±0.37	0.72±0.52
GSH	1.23±1.14	**0.78**±0.63	0.87±0.80	0.86±0.62	1.09±0.98
Glc	4.79±4.94	**2.98**±3.07	3.57±3.71	3.12±3.26	4.01±5.03
Gln	7.55±6.40	4.47±4.19	4.61±4.14	4.07±3.77	**3.79**±4.48
Glu	0.48±0.32	**0.42**±0.29	0.56±0.29	0.46±0.30	0.57±0.33
Gly	2.01±1.87	1.32±1.30	1.31±1.45	**1.25**±1.16	1.33±1.38
Ins	**0.44**±0.31	0.46±0.29	0.48±0.30	0.50±0.30	0.60±0.38
Lac	105.51±59.56	35.25±29.96	29.69±24.86	**28.87**±26.99	30.31±42.18
NAA	0.36±0.26	**0.28**±0.24	0.36±0.28	0.32±0.27	0.35±0.28
NAAG	4.95±3.83	2.27±2.45	2.01±2.51	**1.99**±2.32	2.29±2.85
PCho	1.46±1.62	1.18±1.12	**1.06**±1.00	1.37±1.26	1.49±1.62
PCr	0.51±0.39	**0.44**±0.30	0.52±0.30	0.48±0.30	0.53±0.36
PE	2.30±2.09	**1.91**±1.76	2.13±1.89	2.19±1.92	2.43±2.48
Tau	1.31±1.38	**1.03**±0.73	1.08±0.74	1.07±0.70	1.68±1.87
sIns	1.59±1.60	1.35±1.24	**1.30**±1.08	1.38±1.19	1.57±1.75
Cr+PCr	0.27±0.23	**0.22**±0.17	0.35±0.23	0.25±0.18	0.33±0.24
GPC+PCho	0.52±0.55	0.40±0.33	**0.40**±0.31	0.47±0.35	0.70±0.59
Glc+Tau	1.66±1.73	**1.07**±0.85	1.18±0.97	1.10±0.84	1.90±2.10
Glu+Gln	0.61±0.48	**0.32**±0.27	0.37±0.29	0.33±0.27	0.49±0.36
Ins+Gly	0.38±0.34	**0.28**±0.21	0.35±0.26	0.32±0.24	0.47±0.37
NAA+NAAG	0.47±0.25	**0.19**±0.15	0.27±0.20	0.22±0.18	0.26±0.20

Table A.2: Table with the mean and variance of the relative error obtained from the short echo-time tests with 0.3 noise dataset.

Metabolite	SE-f	SE-t	SE-TF	SE-TP	SE-g	SE-subT
Ala	0.16±0.11	0.20±0.14	1.26±1.02	0.31±0.27	**0.16**±9.53e-02	0.20±0.16
Asc	0.17±0.11	0.21±0.15	0.77±0.71	0.30±0.26	**0.16**±9.69e-02	0.22±0.16
Asp	0.16±0.11	0.25±0.18	1.45±1.07	0.32±0.27	**0.16**±9.24e-02	0.22±0.18
Cr	0.19±0.15	0.24±0.17	1.48±1.17	0.44±0.37	**0.16**±9.49e-02	0.26±0.21
GABA	0.16±0.10	0.23±0.16	1.64±1.22	0.31±0.26	**0.16**±9.52e-02	0.23±0.16
GPC	0.17±0.13	0.36±0.25	0.43±0.43	0.35±0.29	**0.16**±9.82e-02	0.23±0.18
GSH	0.16±0.11	0.29±0.20	0.97±0.91	0.30±0.26	**0.16**±9.79e-02	0.21±0.17
Glc	0.16±0.11	0.19±0.12	1.20±0.91	0.31±0.26	**0.15**±9.51e-02	0.20±0.16
Gln	0.16±0.11	0.22±0.14	1.48±1.21	0.31±0.26	**0.16**±9.76e-02	0.22±0.16
Glu	0.19±0.14	0.42±0.23	3.36±3.38	0.51±0.41	**0.15**±9.62e-02	0.50±0.41
Gly	0.16±0.11	0.29±0.19	1.61±1.19	0.30±0.24	**0.16**±9.73e-02	0.21±0.16
Ins	0.21±0.17	0.31±0.19	3.93±2.01	0.48±0.49	**0.15**±9.23e-02	0.47±0.43
Lac	0.16±0.11	0.19±0.11	1.26±0.83	0.30±0.27	**0.16**±9.65e-02	0.20±0.16
NAA	0.19±0.15	0.31±0.23	3.15±1.50	0.76±0.58	**0.16**±9.81e-02	0.64±0.57
NAAG	0.16±0.11	0.19±0.13	1.32±1.05	0.32±0.28	**0.15**±9.10e-02	0.21±0.19
PCho	0.16±0.11	0.18±0.12	1.20±0.84	0.32±0.28	**0.15**±9.51e-02	0.21±0.16
PCr	0.20±0.17	0.24±0.17	1.80±1.35	0.32±0.28	**0.16**±9.25e-02	0.35±0.30
PE	0.16±0.11	0.19±0.13	1.52±1.05	0.32±0.29	**0.16**±9.55e-02	0.21±0.17
Tau	0.16±0.11	0.35±0.28	1.67±1.12	0.38±0.31	**0.16**±9.84e-02	0.23±0.18
sIns	0.16±0.11	0.20±0.15	1.26±0.91	0.33±0.27	**0.16**±9.58e-02	0.21±0.16
Cr+PCr	0.16±0.13	0.20±0.13	1.62±1.20	0.32±0.27	**0.11**±7.85e-02	0.27±0.22
GPC+PCho	0.13±0.10	0.28±0.20	0.61±0.53	0.31±0.26	**0.12**±7.93e-02	0.19±0.15
Glc+Tau	0.13±8.68e-02	0.26±0.20	1.52±1.02	0.34±0.28	**0.12**±8.07e-02	0.20±0.16
Glu+Gln	0.17±0.13	0.39±0.22	3.17±3.15	0.48±0.39	**0.14**±8.67e-02	0.46±0.37
Ins+Gly	0.17±0.13	0.29±0.17	3.34±1.72	0.40±0.38	**0.12**±7.80e-02	0.38±0.33
NAA+NAAG	0.17±0.14	0.28±0.20	2.99±1.42	0.70±0.54	**0.14**±9.01e-02	0.59±0.52

Table A.3: Mean and standard deviation of the relative error of all CNN topologies using the dataset with noise level of 0.3.

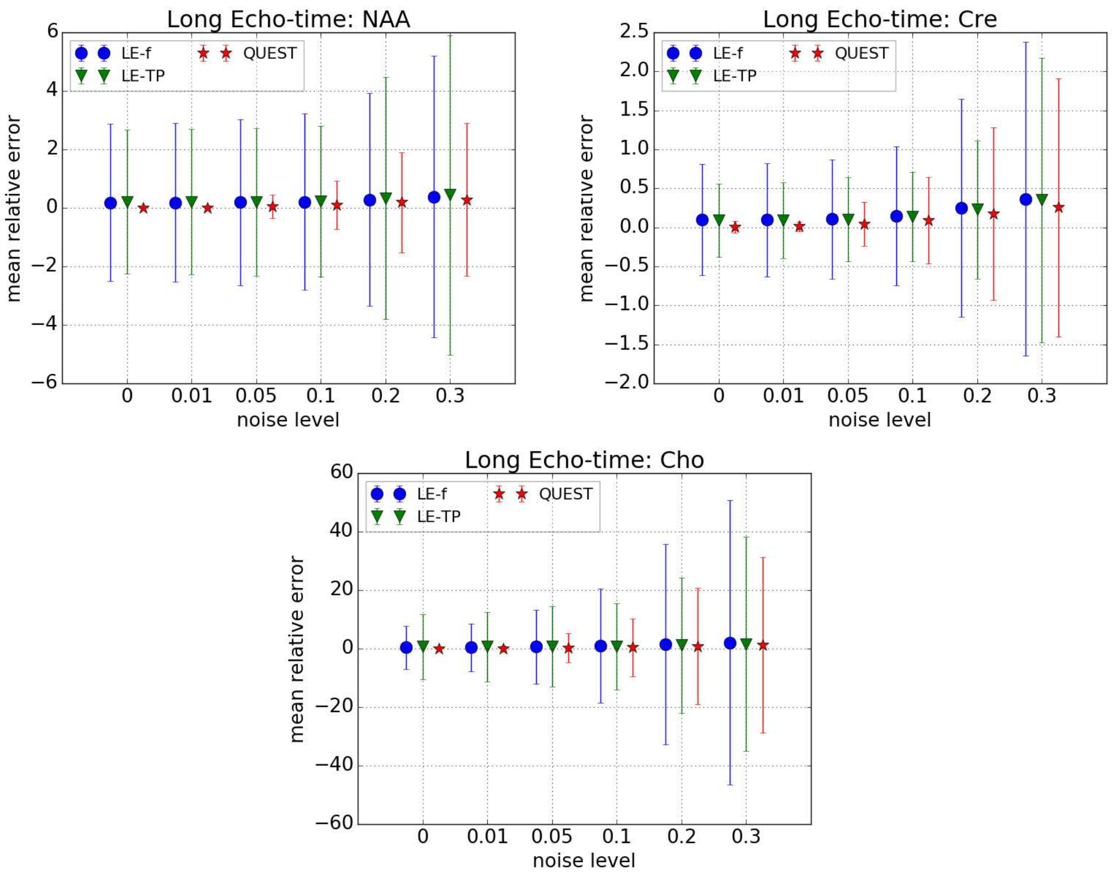

Figure A.1: Plots comparing the standard deviation of the relative error of QUEST and the CNNs *LE-f* and *LE-TP* on the test datasets.

Metabolite	basis	constr	SE-t	SE-f	SE-g
Ala	3.05±3.13	2.52±2.58	0.20±0.14	0.16±0.11	**0.16**±9.53e-02
Asc	2.30±1.97	2.45±2.14	0.21±0.15	0.17±0.11	**0.16**±9.69e-02
Asp	2.94±2.52	3.42±2.83	0.25±0.18	0.16±0.11	**0.16**±9.24e-02
Cr	0.53±0.46	0.56±0.42	0.24±0.17	0.19±0.15	**0.16**±9.49e-02
GABA	3.61±3.63	2.71±2.79	0.23±0.16	0.16±0.10	**0.16**±9.52e-02
GPC	0.52±0.36	0.57±0.37	0.36±0.25	0.17±0.13	**0.16**±9.82e-02
GSH	0.78±0.63	0.86±0.62	0.29±0.20	0.16±0.11	**0.16**±9.79e-02
Glc	2.98±3.07	3.12±3.26	0.19±0.12	0.16±0.11	**0.15**±9.51e-02
Gln	4.47±4.19	4.07±3.77	0.22±0.14	0.16±0.11	**0.16**±9.76e-02
Glu	0.42±0.29	0.46±0.30	0.42±0.23	0.19±0.14	**0.15**±9.62e-02
Gly	1.32±1.30	1.25±1.16	0.29±0.19	0.16±0.11	**0.16**±9.73e-02
Ins	0.46±0.29	0.50±0.30	0.31±0.19	0.21±0.17	**0.15**±9.23e-02
Lac	35.25±29.96	28.87±26.99	0.19±0.11	0.16±0.11	**0.16**±9.65e-02
NAA	0.28±0.24	0.32±0.27	0.31±0.23	0.19±0.15	**0.16**±9.81e-02
NAAG	2.27±2.45	1.99±2.32	0.19±0.13	0.16±0.11	**0.15**±9.10e-02
PCho	1.18±1.12	1.37±1.26	0.18±0.12	0.16±0.11	**0.15**±9.51e-02
PCr	0.44±0.30	0.48±0.30	0.24±0.17	0.20±0.17	**0.16**±9.25e-02
PE	1.91±1.76	2.19±1.92	0.19±0.13	0.16±0.11	**0.16**±9.55e-02
Tau	1.03±0.73	1.07±0.70	0.35±0.28	0.16±0.11	**0.16**±9.84e-02
sIns	1.35±1.24	1.38±1.19	0.20±0.15	0.16±0.11	**0.16**±9.58e-02
Cr+PCr	0.22±0.17	0.25±0.18	0.20±0.13	0.16±0.13	**0.11**±7.85e-02
GPC+PCho	0.40±0.33	0.47±0.35	0.28±0.20	0.13±0.10	**0.12**±7.93e-02
Glc+Tau	1.07±0.85	1.10±0.84	0.26±0.20	0.13±8.68e-02	**0.12**±8.07e-02
Glu+Gln	0.32±0.27	0.33±0.27	0.39±0.22	0.17±0.13	**0.14**±8.67e-02
Ins+Gly	0.28±0.21	0.32±0.24	0.29±0.17	0.17±0.13	**0.12**±7.80e-02
NAA+NAAG	0.19±0.15	0.22±0.18	0.28±0.20	0.17±0.14	**0.14**±9.01e-02

Table A.4: Table comparing QUEST and Deep Learning approaches using 0.3 noise dataset.

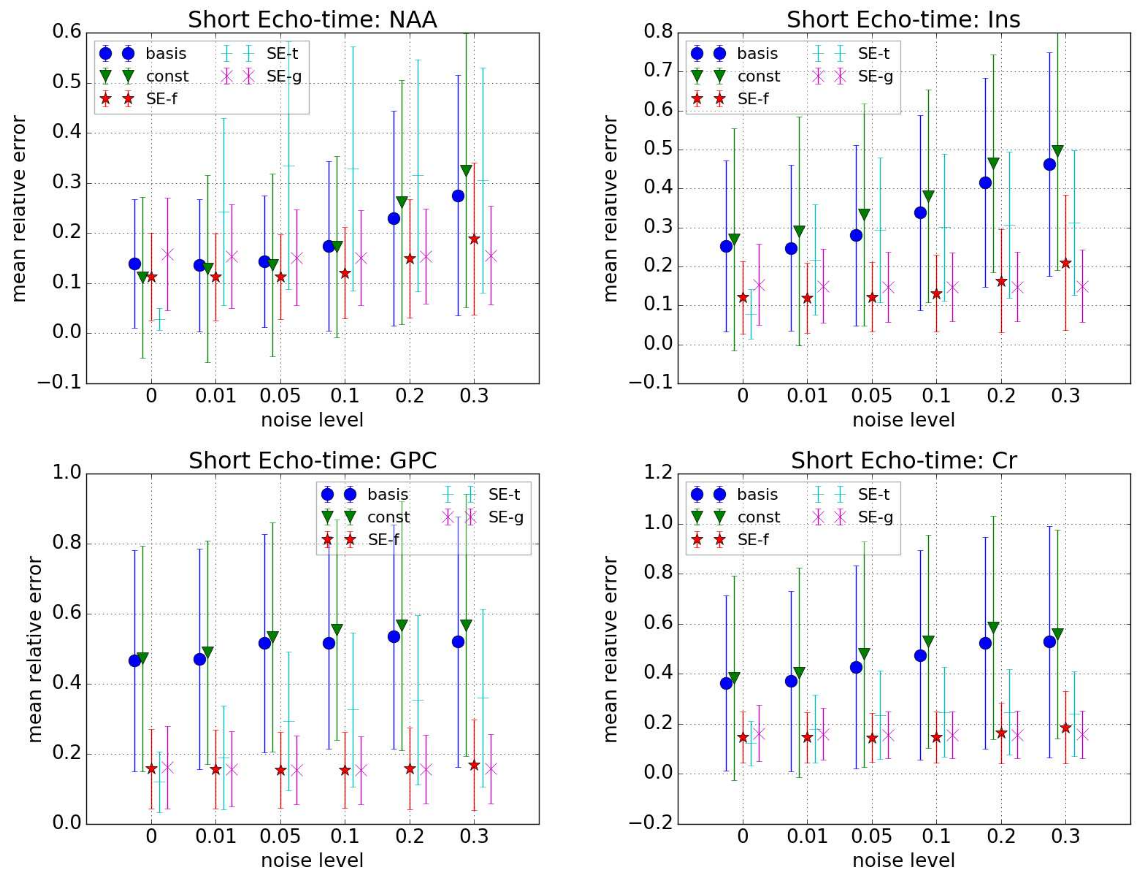

Figure A.2: Plots comparing the standard deviation of the relative error of QUEST approaches *basis* and *const*, and the CNNs *SE-f*, *SE-t* and*SE-g* using the metabolites NAA, Ins, GPC and Cr.

Metabolite	SE-f	SE-t	SE-TF	SE-TP	SE-g	SE-subT
Ala	0.16±0.12	0.15±0.10	**0.15**±0.10	0.15±0.10	0.16±0.11	0.16±0.11
Asc	0.17±0.12	**0.16**±0.10	0.16±0.10	0.16±0.11	0.17±0.12	0.17±0.11
Asp	0.16±0.11	**0.15**±0.10	0.15±9.86e-02	0.15±0.10	0.16±0.11	0.16±0.10
Cr	0.15±0.10	**0.12**±8.86e-02	0.13±9.21e-02	0.13±9.03e-02	0.16±0.11	0.13±9.16e-02
GABA	0.16±0.11	**0.16**±9.77e-02	0.16±9.66e-02	0.16±9.72e-02	0.16±0.11	0.16±0.10
GPC	0.16±0.11	**0.12**±8.62e-02	0.15±0.10	0.14±9.88e-02	0.16±0.12	0.14±9.99e-02
GSH	0.16±0.11	**0.15**±0.11	0.16±0.10	0.16±0.11	0.16±0.11	0.16±0.11
Glc	0.16±0.12	0.16±0.10	**0.16**±0.10	0.16±0.11	0.16±0.11	0.16±0.11
Gln	0.16±0.11	0.15±0.10	0.15±0.10	**0.15**±0.10	0.16±0.11	0.16±0.11
Glu	0.12±9.04e-02	**4.87e-02**±3.94e-02	4.95e-02±3.93e-02	7.01e-02±5.42e-02	0.15±0.11	7.31e-02±5.72e-02
Gly	0.16±0.12	**0.15**±0.10	0.15±0.10	0.15±0.10	0.16±0.11	0.16±0.10
Ins	0.12±9.20e-02	7.83e-02±6.38e-02	7.67e-02±6.23e-02	**7.62e-02**±5.94e-02	0.15±0.10	7.88e-02±6.19e-02
Lac	0.16±0.11	0.16±0.10	**0.16**±0.10	0.16±0.11	0.16±0.11	0.16±0.11
NAA	0.11±8.80e-02	2.87e-02±2.17e-02	**2.85e-02**±2.28e-02	3.28e-02±2.56e-02	0.16±0.11	3.49e-02±2.73e-02
NAAG	0.15±0.11	**0.15**±9.97e-02	0.15±9.84e-02	0.15±9.97e-02	0.15±0.11	0.15±0.10
PCho	0.16±0.11	**0.15**±0.10	0.15±0.10	0.15±0.10	0.16±0.11	0.15±0.11
PCr	0.13±9.23e-02	**0.10**±8.33e-02	0.11±8.40e-02	0.11±8.24e-02	0.16±0.11	0.11±8.50e-02
PE	0.16±0.12	**0.15**±0.10	0.15±0.10	0.15±0.10	0.16±0.11	0.16±0.11
Tau	0.17±0.12	**0.14**±0.10	0.16±0.10	0.15±0.10	0.17±0.11	0.15±0.11
sIns	0.16±0.12	**0.15**±0.10	0.15±0.10	0.16±0.10	0.16±0.11	0.16±0.11
Cr+PCr	8.15e-02±6.20e-02	**5.35e-02**±4.06e-02	5.59e-02±4.19e-02	5.71e-02±4.30e-02	0.11±8.58e-02	5.94e-02±4.38e-02
GPC+PCho	0.12±8.70e-02	**8.69e-02**±6.38e-02	0.11±7.87e-02	0.10±7.42e-02	0.12±9.13e-02	0.10±7.66e-02
Glc+Tau	0.13±9.12e-02	**0.11**±7.76e-02	0.12±8.01e-02	0.11±8.04e-02	0.13±8.99e-02	0.12±8.19e-02
Glu+Gln	0.10±8.08e-02	**4.27e-02**±3.38e-02	4.39e-02±3.41e-02	6.37e-02±4.81e-02	0.14±9.82e-02	6.62e-02±5.10e-02
Ins+Gly	9.56e-02±7.41e-02	**6.17e-02**±4.86e-02	6.30e-02±4.81e-02	6.39e-02±4.93e-02	0.12±8.49e-02	6.46e-02±5.02e-02
NAA+NAAG	0.10±8.04e-02	**2.65e-02**±2.05e-02	2.75e-02±2.10e-02	3.15e-02±2.41e-02	0.15±0.10	3.32e-02±2.56e-02

Table A.5: Mean and standard deviation of the relative error of all CNN topologies using a dataset with 0 noise.

Bibliography

[1] H. Bhat, B. R. Sajja, and P. A. Narayana. Fast quantification of proton magnetic resonance spectroscopic imaging with artificial neural networks. *Journal of Magnetic Resonance*, 183:110–122, November 2006.

[2] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.01852, 2015.

[4] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167, 2015.

[5] ISMRM. ISMRM Workshop on MR Spectroscopy: From Current Best Practice to Latest Frontiers, 2016. http://www.ismrm.org/workshops/Spectroscopy16/.

[6] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[7] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[8] H. Ratiney, M. J. Albers, H. Rabeson, and J. Kurhanewicz. Semi-parametric time-domain quantification of HR-MAS data from prostate tissue. *NMR in Biomedicine*, 23(10), 12/2010 2010.

[9] H. Ratiney, Y. Le Fur, M. Sdika, and S. Cavassila. Short Echo Time H1 Chemical Shift Imaging data quantification in the mouse brain at 11.7T using a constrained parametric macromolecular model. In *ISMRM-ESMRMB Joint Annual Meeting*, Stockholm,Sweden, 2010.

[10] H. Ratiney, M. Sdika, Y. Coenradie, S. Cavassila, D. Van Ormondt, and D. Graveron-Demilly. Time-Domain Semi-Parametric Estimation Based on a Metabolite Basis Set. *Nuclear Magnetic Resonance in Biomedicine*, 18:1–13, 2005. article.

[11] Sho Sonoda and Noboru Murata. Neural Network with Unbounded Activations is Universal Approximator. *CoRR*, abs/1505.03654, 2015.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *CoRR*, abs/1409.4842, 2014.

[14] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *CoRR*, abs/1512.00567, 2015.

[15] The HDF Group. Hierarchical Data Format, version 5, 1997-2017. http://www.hdfgroup.org/HDF5/.