Universidade Federal
do Rio de Janeiro
Escola Politécnica

Sound Pressure Estimation Method Using Traffic Cameras and Convolutional Neural Networks

Matheus Silva de Lima

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores:
José Gabriel Rodriguez Carneiro Gomes
Pedro de Carvalho Cayres Pinto

Rio de Janeiro
Fevereiro de 2021

Sound Pressure Estimation Method Using Traffic Cameras and Convolutional Neural Networks

Matheus Silva de Lima

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Matheus Silva de Lima

Orientador:

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph.D.

Orientador:

Pedro de Carvalho Cayres Pinto

Pedro de Carvalho Cayres Pinto, M.Sc.

Examinador:

Prof. Julio Cesar Boscher Torres, D.Sc.

Examinador:

Mariane R. Petraglia

Prof. Mariane Rembold Petraglia, Ph.D.

Rio de Janeiro

Fevereiro de 2021

Declaração de Autoria e de Direitos

Eu, *Matheus Silva de Lima* CPF *161.992.327-04*, autor da monografia *Sound Pressure Estimation Method Using Traffic Cameras and Convolutional Neural Networks*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.
2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e idéias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.
3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.
4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.
5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.
6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.
7. Por ser verdade, firmo a presente declaração.

Matheus Silva de Lima

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es).

DEDICATÓRIA

Para papai, mamãe e meu irmão.

AGRADECIMENTO

Agradeço a luta do povo Brasileiro, que em meio a tantas intempéres persiste, e na educação, pública, gratuita e da mais alta qualidade, encontra sua voz já rouca, mas em alto e bom tom.

Agradeço ao CNPq pela bolsa PIBIC oferecida ao projeto, e a todo o suporte prestado durante toda a minha graduação. Agradeço aos meus orientadores, José Gabriel e Pedro Cayres por todo a orientação feita durante esse projeto.

Agradeço a meus pais, Manoel e Maria. É graças a vocês dois e tudo que batalharam que posso ser quem sou. Minha gratidão é tamanha que não caberia mesmo em todas as páginas desse trabalho. Agradeço ainda meu irmão Thiago, que sempre foi meu exemplo. Agradeço a toda minha família, Adalberto Santana, Nilda Ferreira, Ana Maria da Silva, Jéssica Lisla, Ana Carolina, Ana Clara, Sophia, Tia Antônia e Tio Sérgio, entre tantas outras pessoas maravilhosas.

Agradeço às minhas amizades de já incontáveis anos: Alexandre Wanick, Bruno Mesquita, Igor Cantelmo, Vitória Ribeiro, Letícia Freitas, Fernanda Moraes, Sarah Medeiros, Nilton Dias, Leonardo Guilhon, Gregório Queiroz, Gabriel Chezorim. Agradeço a todas as amizades feitas durantes esses anos de faculdade, e que viveram de perto nossa batalha: Vinícius Mello, Andressa Vasconcelos, Charles *charlinho* Alexandre, Letícia Ecard, Fabiana Ferreira, Lucas Cerqueira, Gabriel *padrinho* Morgado, Igor Bandeira, Alan Carpilovsky, Caroline Coelho, Marina Lacerda, dentre muitas outras pessoas que tive o prazer de conhecer durante esses anos.

Agradeço em especial a pessoa que durante muito tempo foi minha família, Alyssa Takazume. Ninguém acompanhou tão próximo essa trajetória, nem me deu tanto apoio, e nem mudou tanto minha vida quanto ela. Agradeço ainda a Rosimery Takazume por todo o apoio que me foi dado durante toda a graduação.

Agradeço a todos os amigos que fiz na Escola Modelo de Língua Japonesa do Rio de Janeiro: Matsudaira Sensei, Rafael Longui, Rika Sensei, Juliana Sensei, Felipe Sensei e em especial, Jorge Alison, que foi o responsável por organizar toda a minha papelada de intercâmbio, e que acabou tornou-se um grande amigo.

私のもう一つの国、日本で会った皆様に心から感謝の気持ちを伝えたいと思います。福岡で会った山口さん、九州大学のスタッフOiwaさんと由紀子さ

ん、神戸で会った鈴木さん、西山さん、琴美さんと酒井さん。今後お会いすることはできないかもしれませんが、皆さんとの出会いを通して、私は大きく成長することができました。皆さんは私の心の支えです。ありがとうございました。

Por fim, um enorme agradecimento a todos os professores da UFRJ que me ensinaram todo o conhecimento científico e ético acerca da profissão de engenheiro e suas responsabilidades. Mais que professores, muitos de vocês são verdadeiros amigos, e que carregarei comigo por toda a vida ♡

RESUMO

O nível de pressão sonora em ruas é uma informação importante para o planejamento urbano. Medir esta informação através de uma rede de microfones é uma solução que requer a implementação de uma nova infraestrutura, o que pode demandar um investimento monetário considerável, além da necessidade de calibração rotineira dos sensores. Cidades costumam já possuir uma infraestrutura de câmeras de tráfego. Tais câmeras entretanto não costumam possuir microfones. Neste trabalho é proposto uma solução para avaliação de pressão sonora em cidades utilizando uma infraestrutura de câmeras de tráfego, estimando a pressão sonora a partir de suas imagens.

Foram testados modelos de redes neurais convolucionais com arquitetura não-temporal e propostos modelos com arquitetura temporal (LSTM, do inglês "*long short-term memory*"). Utilizando uma base de dados de 38 vídeos de tráfego com áudio e um total de 995 minutos, foram treinados 130 variações de redes convolucionais para fazer a predição de valores médios do sinal de áudio a partir de imagens do vídeo. O desempenho das redes neurais foi avaliado em termos do erro médio quadrático entre as suas saídas e os seus alvos, e também em termos da correlação entre esses sinais, fazendo uma validação cruzada entre 10 diferentes "*folds*".

Neste trabalho foi observado que as redes neurais temporais não-causais baseadas em LSTM obtêm consistentemente resultados melhores que aquelas que não possuem arquitetura temporal. As redes propostas obtiveram um erro de medição abaixo das estudadas em trabalhos anteriores, demonstrando uma correlação entre o sinal predito e o real de 71,3%. As redes LSTM também apresentam um sinal de saída menos ruidoso que aquele apresentado pelas redes não-temporais. O uso de técnicas regularizadoras se mostra decisivo para o treinamento. A rede convolucional testada que apresentou o melhor resultado foi a VGG16.

Foi concluído que a predição do nível sonoro de ruas a partir de imagens de câmeras é possível. Foi constatado que o uso de redes classificadoras auxiliares (como a *Faster R-CNN*) têm potencial para melhorar as predições.

Palavras-Chave: Pressão sonora, aprendizado de máquina, redes convolucionais, processamento audiovisual, cidades inteligentes.

ABSTRACT

Sound pressure level on streets is an important information for urban planning. Measurement of this information using microphones requires a new infrastructure, what may require a considerable monetary investment. Cities usually have a CCTV (closed-circuit television) infrastructure to monitor traffic, but such cameras do not have microphones. A solution for sound pressure inference using CCTV cameras, estimating sound pressure on streets from traffic images, is proposed.

This work tests previously proposed non-temporal convolutional neural networks architectures, and compare them with a proposed temporal convolutional neural network based on LSTM (long-short therm memory) architecture. The database used is composed of 38 videos showing a traffic intersection over different days, hours and weather conditions, with overall length of 995 minutes. Using the images and sound signal from such videos, training of 130 model variations are conducted in order to predict the mean sound pressure on unseen data, using only the video images as input. Performance evaluation was made using the mean squared error and Pearson correlation of the predicted and targeted output signals, cross validating with 10 different folds.

It was observed that LSTM based neural networks consistently yield better results compared to non-temporal based architectures. The proposed neural networks had estimation errors below previously proposed networks, with a correlation of 71.3% between predicted and target signals. Regularization methods were essential during training. The convolutional neural network that yielded the best results was the VGG16. Classification architectures such as the Faster R-CNN have significant potential for improving prediction results.

It was concluded that traffic sound pressure prediction from CCTV camera images is possible within an error limit. For future works, improvements such as creating a new database with different places and better audio capture, exploring 3D convolutions, convolutional-LSTM layers, and investigating how classification networks may further improve results are proposed.

Key-words: Sound pressure, machine learning, convolutional neural networks, audiovisual signal processing, smart cities.

Terms and Acronyms

UFRJ - Universidade Federal do Rio de Janeiro.

CCTV - Closed-circuit Television.

AI - Artificial Intelligence.

SGD - Stochastic Gradient Descent.

Adam - Adaptive Moment Estimation.

CNN - Convolutional Neural Network.

VGG - Visual Geometry Group.

FC - Fully Connected (layer).

GMP - Global Max Pooling.

GAP - Global Average Pooling.

ResNet - Residual Network.

ILSVRC - ImageNet Large Scale Visual Recognition Challenge.

COCO - Common Objects in Context.

RNN - Recurrent Neural Network.

LSTM - Long-short Term Memory.

RAM - Random Access Memory.

CPU - Central Processing Unit.

GPU - Graphics Processing Unit.

PSD - Power Spectrum Density.

MSE - Mean Squared Error.

R-CNN - Region-based Convolutional Neural Network

Contents

List of Figures **xiii**

List of Tables **xvi**

1 Introduction **1**

- 1.1 Theme 1
- 1.2 Delimitation 2
- 1.3 Justification 2
- 1.4 Objectives 2
- 1.5 Methodology 3
- 1.6 Text Structure 3

2 Theory **5**

- 2.1 Theoretical Background 5
  - 2.1.1 A Brief Introduction to Machine Learning 5
  - 2.1.2 Classification and Regression Model 6
  - 2.1.3 Model Training in Supervised Learning 7
- 2.2 Convolutional and Recurrent Neural Networks 11
  - 2.2.1 VGG16 11
  - 2.2.2 ResNet50 16
  - 2.2.3 Inception-V3 18
  - 2.2.4 Long Short-term Memory 19
  - 2.2.5 Transfer Learning 22
- 2.3 Data Processing 23
  - 2.3.1 Pre-Processing: Data Normalization 24
  - 2.3.2 Post-Processing: Moving Average and Correlation Coefficient . 25

3 Methodology 27
3.1 Structure and Implementation 27
3.1.1 Dataset Generation 27
3.1.2 Data Pre-Processing 29
3.1.3 Fold Generation 31
3.2 Models 33
3.2.1 Base Models 33
3.2.2 Non-LSTM based models 33
3.2.3 LSTM Based Models 34
3.2.4 Models with Auxiliary Inputs 36
3.3 Model Evaluation 38

4 Results 39
4.1 Hyperparameter Choice 40
4.1.1 Reported Results for First Training 40
4.2 Cross-fold Training 43
4.2.1 Results for non-LSTM Models 43
4.2.2 Influence of auxiliary input on non-LSTM models 48
4.2.3 Results for LSTM models 49
4.2.4 Influence of auxiliary input on LSTM models 53
4.3 Regularization on Temporal and Non-Temporal Models 55
4.4 Ablation study: LSTM vs non-LSTM models 57
4.4.1 Model and training complexity 57
4.4.2 MSE and Correlation differences 58
4.4.3 Time and frequency response 59
4.5 Window Length Analysis 59

5 Conclusion 62
5.1 Future Works 63

Bibliography 64

A Detailed table of folds 67

B Detailed table of trained networks 69

List of Figures

2.1 Figure shows: (a) illustration of training process; (b) illustration of evaluation process. 10

2.2 Convolution of an 8×8 gray-scale image and a 3×3 kernel filter. . . . 12

2.3 The convolution of an 8×8 image and a 3×3 kernel results in a 6×6 image, as both the first and last rows/columns of the input image cannot be calculated. 12

2.4 In a low resolution image, the loss of two columns and rows is noticeable. When the input is larger, however, this is not a problem. The VGG16 is trained in color images of tensor shape 224×224×3. 13

2.5 Examples of other 3×3 kernels applied to the same 8×8 original image. 13

2.6 The VGG16 architecture uses a 2×2 max pooling with stride 2. It reduces the output to 1/4 the input size. 15

2.7 The VGG16 structure is composed of two parts: the convolution part, also called feature extractor; and the fully connected part, also called classifier (see Section 2.2.5). 16

2.8 In a residual network, output $\hat{y} = f(\bar{x}, \bar{w}) + \bar{x}$. It is hypothesized that training $f(\bar{x}, \bar{w}) = \hat{y} - \bar{x}$ is an easier task than training $f(\bar{x}, \bar{w}) = \hat{y}$. . 17

2.9 ResNet34 architecture. The ResNet50 network follows the same principle as the ResNet34, but its depth is extended to 50 layers. 17

2.10 Inception-V3 overall architecture. 19

2.11 Translation is a heavily time and context dependent process. Translating word by word is not an option. 20

2.12 RNNs have an internal loop that allows information to be represented and to remain available for computation. This way they can be used to process temporal information. 20

2.13 RNNs loop unrolled for each time step. It is clear that information at $t = 0$ affects output at $t = 2$. 20
2.14 LSTM cell structure. 22
2.15 Example of FC layer structure. 23
2.16 Process of Gaussian normalization. 24
2.17 Moving average frequency response for $L = 2$. Graph of $\omega \to |H(e^{j\omega})|$. For the maximum frequency $\omega = \pi$, $|H(e^{j\omega})| = 0$. 26

3.1 Video extraction process. 29
3.2 Sound pressure calculation. 30
3.3 Data generation: (a) how a single video is processed. In this stage, three files are generated, containing the images, targets and the videos mean and standard deviation; (b) how the models input and output files are generated. All video images are concatenated and normalized into a single file. Targets are also concatenated, but are not normalized. 32
3.4 Non-LSTM based models. 33
3.6 Dataset loading. 34
3.5 LSTM based models. 35
3.7 Auxiliary input models. 37
3.8 Faster R-CNN features tensor. 38

4.1 Training loss over 90 epochs. Loss plot over epochs present larger spikes when using GMP: (a) GAP training loss; (b) GMP training loss. 41
4.2 Difference between using LSTM and not using LSTM layer: (a) LSTM models training loss and time prediction; (b) non-LSTM models training loss and time prediction. 42
4.3 Difference between using L2 regularizer and no regularizer: (a) LSTM with L2 regularization; (b) LSTM with no regularization. 42
4.4 Comparison between using and not using a hidden FC layer on non-LSTM models (VGG16). Clear lines are the training curves for individual folds. It is noticeable that the hidden-layer makes the model learn faster and better at the initial epochs, but it easily over-fits on the following epochs. 45

4.5 VGG16 non-LSTM time predictions with and without the hidden FC layer. Results from the last-epoch of training on fold 4. 46
4.6 Comparison between VGG16, ResNet50 and InceptionV3 based non-LSTM models with hidden FC layer and last-epoch time prediction on fold 4. We can see the effects of overfitting the hidden-layer has in these networks. 47
4.7 Best individual models for LSTM networks without an auxiliary input. Predictions over fold 4, best epoch. 51
4.8 Time prediction for model 22 on fold 0. We can see that this model did not converge. 53
4.9 Model 60: Best LSTM model with L2 regularization over 300 epochs. 55
4.10 Model 61: Best LSTM model with dropout regularization over 300 epochs. 56
4.11 Model 62: Best non-LSTM model with L2 regularization over 300 epochs. 56
4.12 Model 63: Best non-LSTM model with dropout regularization over 300 epochs. 56
4.13 LSTM and non-LSTM base models structure (VGG16 extractor). . . 58
4.14 Comparison of best non-LSTM and LSTM, without auxiliary input models. Image show the time response and power spectrum density of the signals. 59
4.15 Predictions behaviour over different time windows for best LSTM model (model 61). Time predictions on fold 4. 61

5.1 Detailed view of the best single-input model. 62

List of Tables

3.1 Detailed list of videos in the available dataset. 28

4.1 10 best individual results for non-LSTM networks. 44
4.2 non-LSTM mean MSE, ordered by best epoch MSE. 44
4.3 Mean correlation for non-LSTM models. 48
4.4 Mean MSE for all non-LSTM models, including models with an auxiliary input. 49
4.5 Best individual models for LSTM networks without an auxiliary input. 50
4.6 LSTM models with best mean MSE, without an auxiliary input. . . . 50
4.7 LSTM models ordered by mean best epoch correlation. 52
4.8 LSTM models ordered by best epoch mean MSE. 54
4.9 LSTM models ordered by best epoch mean correlation. 54
4.10 Best non-LSTM and LSTM models MSE, validation data. 57
4.11 Best non-LSTM and LSTM models correlation, validation data. . . . 57
4.12 Comparison of best non-LSTM and LSTM, without auxiliary input models. 58
4.13 MSE behaviour over different time windows. 60
4.14 Correlation behaviour over different time windows. 60

A.1 Table of videos per fold 67
A.2 Table of fold hours 68

B.1 Table of trained networks in multiple folds 69
B.2 Table of trained networks in one fold 70

Chapter 1

Introduction

In this chapter, the reader is introduced to the research topic. That is, what the object of study is, what the targeted problem is, and which methods are used for proposing a solution. Finally, a brief explanation of the structure of this text is given.

1.1 Theme

This research is about the estimation of noise pollution in cities. Noise pollution is a type of environment pollution caused by, among other things, road traffic, railways, air traffic, construction sites *et cetera*. The increase of noise pollution is a prominent characteristic of urbanization worldwide. Previous research efforts show that noise pollution in cities directly impacts the physical and mental health of citizens [1, 2, 3], children development [4] and surrounding wild life [5]. As the awareness about the negative effects of noise pollution arises, so does the necessity of estimating noise levels on cities.

Studies have been conducted worldwide for estimating and monitoring sound levels using statistical inference models [6]. Is such studies, however, manual noise measurements are conducted, which do not translate into a real-time measurement system. It is well known that the most important source of noise pollution in cities is traffic [7], which is already monitored by closed-circuit television systems (CCTV). However, such CCTV cameras usually do not have or cannot legally record audio, in order to enforce privacy rules. This study therefore explores how image and audio information are related in the CCTV systems, and how noise levels can be estimated on streets using only the image information from the CCTV systems.

1.2 Delimitation

This method of noise estimation is limited, as it only accounts for noise produced by objects present within the recorded frame. As such, this method may not satisfactory work close to airport, railway or construction regions. Estimation of noise from images can also be very inaccurate, specially for individual vehicles and uncommon events. For those estimates, one should have training data acquired from properly placed microphones. The proposed method does not, therefore, substitute a precise and generic noise measurement system. This study aims however to predict the average sound pressure intensity on roads and streets, under common traffic and weather conditions.

1.3 Justification

Monitoring of noise distribution on cities allows the development of indexes about quality of life on neighborhoods. It is an important tool for city management and law making. It helps politicians and communities to assert the effectiveness of laws regarding the reduction of noise in city areas. It is also an important criterion for social and real estate development. It can be used in multi-criteria analysis, for determining the optimal location of a new hospital, school or social housing facilities [8].

1.4 Objectives

The general objective of this study is to expand previous research [9] on the topic of image processing applied to CCTV systems for street noise prediction. The specific objective is to reproduce the results from model architectures based on still images (referred to as "non-temporal" model architectures in this work), to introduce temporal model architectures for this problem, identify their optimal hyperparameter configurations, and to compare them within an expanded dataset, using cross-validation methods.

This study can be regarded as successful, because the proposed temporal model architectures yield better prediction results than the previously proposed non-temporal based architecture.

1.5 Methodology

To train deep-learning models for sound pressure prediction using only video frames as an input, a dataset composed by 38 video files with an overall length of 995 minutes was used. All videos were recorded from the same place and angle, approximately 50 meters away from the scene, in different week days, at different morning, afternoon, and night times, and under different weather and illumination conditions.

The sound pressure information for each frame was calculated using the captured sound as targets for our sound pressure prediction models. These videos are divided into 10 different folds, as shown in Appendix A.

Models were then compared in terms of their mean squared error and correlation coefficients, computed over all folds, between the predicted and target sound pressure values.

1.6 Text Structure

In Chapter 2, the theoretical background behind this study is presented. This includes concepts from the broader area of machine learning, a study of models used in this work and some brief concepts of data analysis. The key question is: “what are the fundamentals of our analysis?”.

In Chapter 3, the methodology used in our research is presented. This includes what data was processed, how it was processed and how evaluation was conducted. The key question is: “how to apply theory in order to propose a solution to the problem?”.

In Chapter 4, several results are presented.

Finally, in Chapter 5, this study is ended with our conclusion and some final thoughts on future research ideas about this topic.

Chapter 2

Theory

In this chapter, the main theory points that were used in this work are discussed. It is organized in three sections:

- Theoretical Background
- Convolutional and Recurrent Neural Networks
- Data Processing

2.1 Theoretical Background

In this first section, a brief introduction to the field of machine learning is presented to the reader. Topics about the basics of machine learning and neural networks are discussed.

2.1.1 A Brief Introduction to Machine Learning

Machine learning is a subset of the wider field of artificial intelligence. It deals with algorithms that automatically improve through experience, obtained during the so called *training process* of the algorithm. Given a specific task, the training process consists of trying to interactively improve on how to perform it. There are three main approaches to this, which are:

- Supervised Learning

The algorithm is given a set of labeled data, and is asked to learn a general rule between data and labels. It is similar to a "question and answer" sheet for a test. From the known data, the algorithms learn to generalize and correctly answer questions that have not been previously presented to them.

- Unsupervised Learning

  The algorithm is given a set of unlabeled data, and tries to identify general patterns within the set. An example of unsupervised learning is data clusterization, which consists in finding groups of similar data.

- Reinforced Learning

  The algorithm dynamically interacts with an environment. By interacting with it, it learns to do what is required. For example, an algorithm that learns to park a car.

Each of these methods is better suited to solve different problems. They can be used independently, or together. In this work, given problem nature, the supervised learning method was used.

2.1.2 Classification and Regression Model

In a supervised learning algorithm, it is common to use artificial neural networks.

Artificial neural networks are a class of computer structures inspired on biological neural networks. Although there are other proposed models whose basic units resemble biological neurons more closely [10], artificial neural networks (for now on, only "neural networks") are the most widely used models. Neural network models can be used for non-linear classification or regression.

- Classification is the task of finding the best set of parameters to classify the input. As an example, given pictures of cats and dogs, learning which pictures corresponds to cats and which corresponds to dogs.

- Regression is the task of estimating the relationship between input and output. It is used to predict and forecast possible outcomes. As an example, given a set of points (x_i, y_i), training leads to an approximating function $y = f(x)$ that best represents these points. The simplest regression model is the linear regression, where $f(x) = ax + b$.

Classification and regression are, in fact, very similar operations. Classification can be regarded as regression with discrete (or binary) targets.

2.1.3 Model Training in Supervised Learning

Training a neural network under a supervised learning method consists of fitting the model to the available data.

Let $\bar{x} = [x_0, x_1, ..., x_M]^T$ and $\bar{y} = [y_0, y_1, ..., y_M]^T$ be, respectively, the input and output data. The data are labeled, that is, each input x_i corresponds to a target output y_i, that is, $x_i \rightarrow y_i$.

The $\bar{x}$ and $\bar{y}$ elements association is called a dataset, denoted as $(\bar{x}, \bar{y})$. Let a model be described by a general estimation function $f(.)$ of the input data $\bar{x}$, and some parameters $\bar{w}$, so that

$$\hat{y} = f(\bar{x}, \bar{w}) \tag{2.1}$$

is the estimated output.

The hyperparameters $\bar{w}$ are known as the weights of model $f(.)$. Fitting the model to the data means finding the set of weights $\bar{w}$ that better correlates input and output data. Ideally, $\hat{y} = \bar{y}$. That is, the estimation function $f(.)$ perfectly defines $\bar{y}$ given $\bar{x}$. Finding such ideal weights is usually impossible. In practice, instead of looking for ideal weights, a loss function

$$L(\bar{y}, \hat{y}) = \|\bar{y} - \hat{y}\| \tag{2.2}$$

that measures the distance between data $\bar{y}$ and estimation $\hat{y}$ is defined. This way, finding the optimal set of weights $\bar{w}$ is equivalent to finding the global minimum L_{min} of the loss function $L(.)$

$$L_{min} = \min_{\bar{w}} L(\bar{y}, f(\bar{x}, \bar{w})). \tag{2.3}$$

Finding L_{min} is a hard, non-trivial task. Local minima can however be found with *gradient descent algorithms.* A gradient descent consists of calculating the sequence

$$\bar{w}_{n+1} = \bar{w}_n - \gamma \nabla L(\bar{y}, f(\bar{x}, \bar{w}_n)), \tag{2.4}$$

where ∇ is the gradient operator

$$\nabla h(a_0, a_1, ..., a_N) = \left\langle \frac{\partial h}{\partial a_0}, \frac{\partial h}{\partial a_1}, ..., \frac{\partial h}{\partial a_N} \right\rangle . \tag{2.5}$$

The parameter γ is called learning rate. It defines the algorithm convergence rate. Larger γ leads to faster minimization, but also a larger convergence instability. Given a small enough γ, the sequence $L(\bar{y}, \bar{x}, \bar{w}_0), L(\bar{y}, \bar{x}, \bar{w}_1), ...$ is monotonically decreasing. The loss function is therefore minimized in the neighborhood of the initial $\bar{w}$.

If the dataset is very large, then calculating the gradient for the entire dataset can be very demanding, and so the gradient descent algorithm becomes too costly. Instead, enhanced methods are used. These enhanced methods calculate the gradient by sampling the dataset in smaller partitions called “batches”. Each batch contains a subset of the entire dataset. Sampled data are not repeated among batches. Letting $(\bar{x}_B, \bar{y}_B)$ be the sampled data, some examples of enhanced methods are:

- Stochastic Gradient Descent (SGD)

  This is the gradient descent algorithm applied in batches.

  $$\bar{w}_{n+1} = \bar{w}_n - \gamma \nabla L(\bar{y}_B, f(\bar{x}_B, \bar{w}_n)) \tag{2.6}$$

- SGD with Momentum

  This is the SGD algorithm, but the gradient is filtered with a moving average filter (see Section 2.3.2).

  $$\bar{w}_{n+1} = \bar{w}_n - \gamma \nu_{n+1} \tag{2.7}$$

  $$\nu_{n+1} = \beta \nu_n + (1 - \beta) \nabla L(\bar{y}_B, f(\bar{x}_B, \bar{w}_n)) \tag{2.8}$$

Here, $\beta \in [0, 1)$ is a filter constant (momentum hyperparameter), and ν_n can be seen as the "mean gradient". SGD with momentum helps to filter the gradient noise over successive batches.

- Adaptive Moment Estimation (Adam)

  This is a combination of the SGD with momentum and RMSprop [1] algorithms. It uses two momentum variables: ν is the moving average filter of the gradient (mean gradient), and the scalar ζ is the variance estimate of the gradient.

$$\bar{w}_{n+1} = \bar{w}_n - \gamma\nu_{n+1} \odot (\zeta_{n+1} + \epsilon)^{-1/2} \tag{2.9}$$

$$\nu_{n+1} = \beta_1\nu_n + (1 - \beta_1)\nabla L_B(\bar{w}_n) \tag{2.10}$$

$$\zeta_{n+1} = \beta_2\zeta_n + (1 - \beta_2)[\nabla L_B(\bar{w}_n) \odot \nabla L_B(\bar{w}_n)] \tag{2.11}$$

$$L_B(\bar{w}_n) = L(\bar{y}_B, f(\bar{x}_B, \bar{w}_n)) \tag{2.12}$$

  Here, $\beta_1, \beta_2 \in [0, 1)$ are filter constants (momentum hyperparameters). The scalar $\epsilon > 0$ is a very small constant, to prevent division by zero. The symbol $\odot$ denotes the Hadamard product (point-wise product).

Because of hardware implementation, the batch size in enhanced methods is usually chosen to be a power of 2. Gradient is then estimated from all batches. An epoch ends when every batch in the dataset has been processed.

Using these gradient descent based methods, the minimization of the loss function $L(.)$ is conducted over the dataset $(\bar{x}, \bar{y})$. The model is however expected to perform well on any given, not previously seen, dataset. To evaluate how the model performs on unseen data, the dataset is usually separated into two parts:

- Training Data

  Data actively used in the optimization algorithm. Models are fit to the training data.

- Testing Data

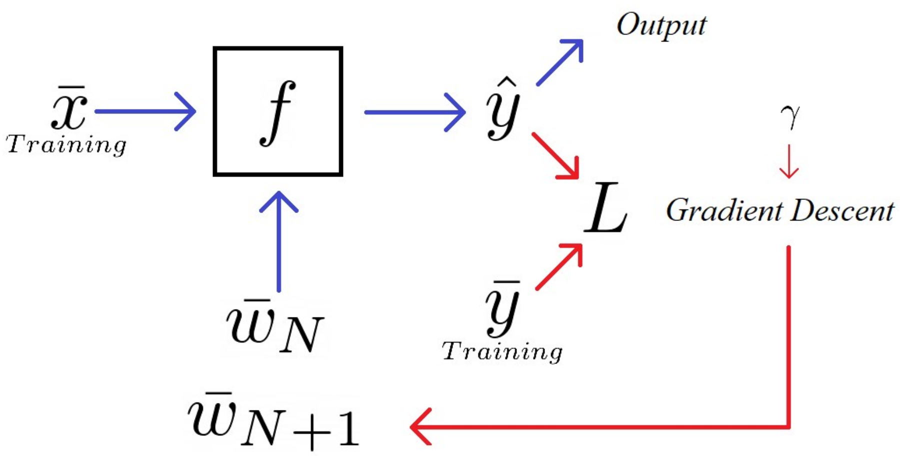
---

[1] Proposed by Geoffrey Hilton in his class "Neural Networks for Machine Learning" [11]

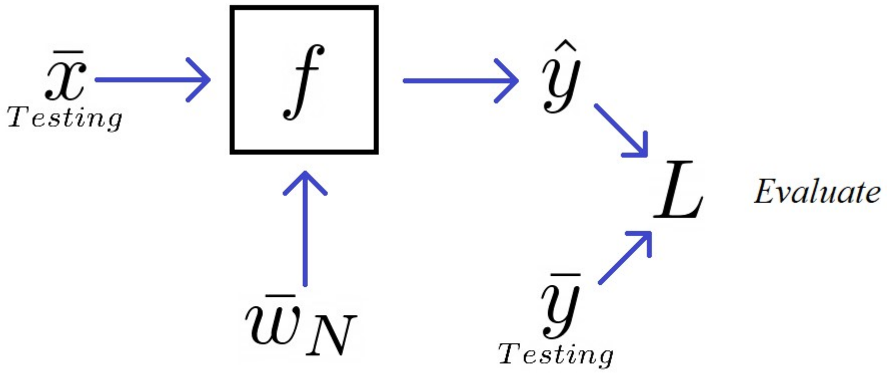
Data not used by the optimization algorithm. Those data are used only for evaluating how the model is performing on untrained data.

Another common procedure when evaluating the model is fitting the model on different folds. A fold is a set of training and test data. The idea is that training and test data vary among the different folds. Evaluating the performance of the model on different folds makes the evaluation less dependent on data.

An overview of the training and evaluation process can be seen on Figure 2.1.



(a) Illustration of training process.



(b) Illustration of evaluation process.

Figure 2.1: Figure shows: (a) illustration of training process; (b) illustration of evaluation process.

2.2 Convolutional and Recurrent Neural Networks

In this section, the concepts of convolutions and recurrent layers on neural networks are introduced, that is, the base models used in this work and the basics of their internal design.

2.2.1 VGG16

The VGG16 is a deep convolutional neural network structure proposed by Karen Simonyan and Andrew Zisserman [12]. The acronym VGG stands for "Visual Geometry Group Laboratory", Oxford University. It was the laboratory entry for the 2014 "ImageNet large scale visual recognition challenge" (ILSVRC2014)

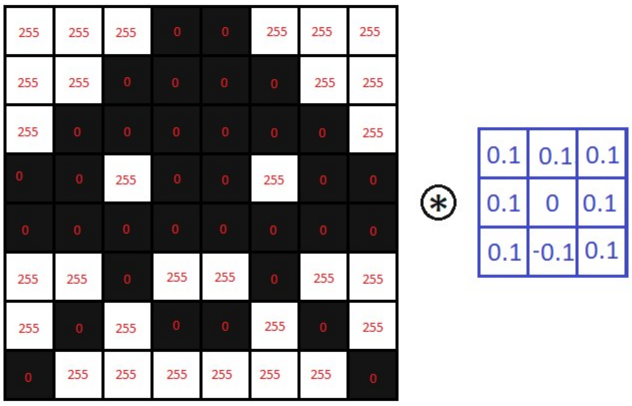
The ILSVRC 2014 was a contest that aimed to "evaluate algorithms for object detection and image classification at large scale" [13]. Teams competed in two categories:

- To detect objects from 200 different categories within an image, on fully labeled data (called "object localization").
- To classify images from 1000 different possible labels (called "image classification").

The proposed VGG model became known for winning first place in the localization challenge, and attaining second place in the classification one. It consists of consecutive stacked 3×3 convolutions and pooling layers over the image, followed by three fully connected (FC) layers.

1. Convolution

A convolution is a linear operator. Given an input signal $input(x)$ and a filter $filter(x)$, it produces a filtered version $output(x) = input(x) \circledast filter(x)$ of the signal in its output. The given signal can be, for example, an audio signal or an image. In the case of images, the convolution operation is applied between matrices, as follows: let the following 8×8 and 3×3 matrices in Figure 2.2 be, respectively, an example image and filter, also known as "kernel filter" or only “kernel”. In a graphical sense, the matrix can be regarded as an eight-bit 8×8 gray-scale image, where 0 represents black and 255 represents white.



255	255	255	0	0	255	255	255
255	255	0	0	0	0	255	255
255	0	0	0	0	0	0	255
0	0	255	0	0	255	0	0
0	0	0	0	0	0	0	0
255	255	0	255	255	0	255	255
255	0	255	0	0	255	0	255
0	255	255	255	255	255	255	0

0.1	0.1	0.1
0.1	0	0.1
0.1	-0.1	0.1

Figure 2.2: Convolution of an 8×8 gray-scale image and a 3×3 kernel filter.

The convolution process consists of adding each element of the image to its neighbors, weighing the element and its neighbors by the kernel values. As the edge of the image does not have neighbors along all directions, it is usually ignored. In that case, the output image is smaller then the input image, as shown in Figure 2.3.

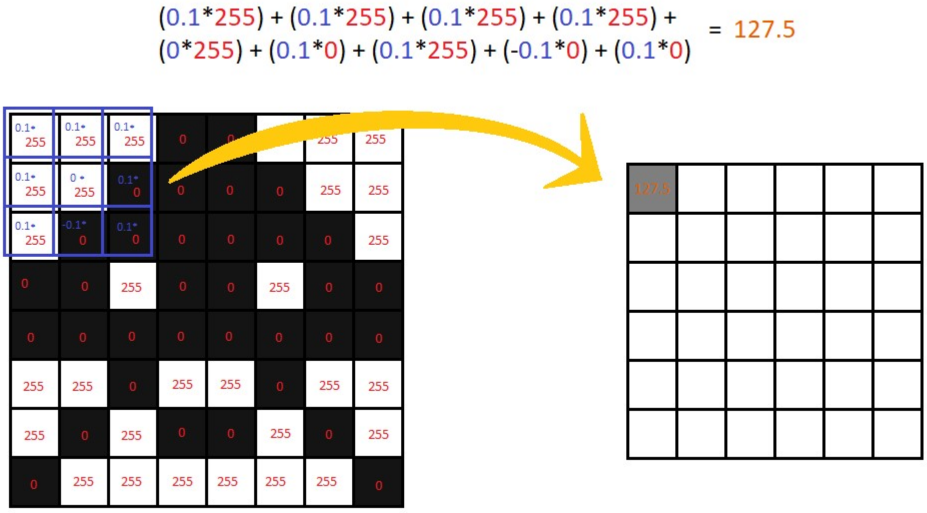


Figure 2.3: The convolution of an 8×8 image and a 3×3 kernel results in a 6×6 image, as both the first and last rows/columns of the input image cannot be calculated.

The operation is then continuously applied to each element of the image until all elements have been calculated, as depicted in Figure 2.4.

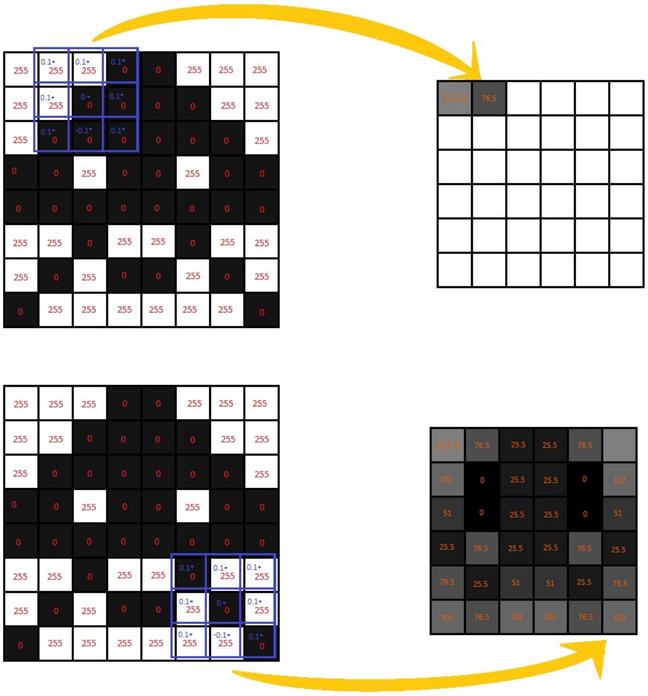
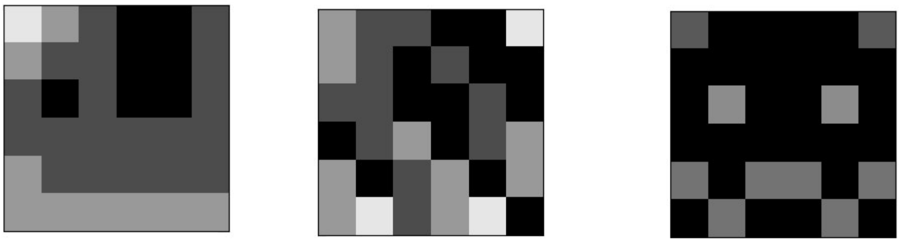


Figure 2.4: In a low resolution image, the loss of two columns and rows is noticeable. When the input is larger, however, this is not a problem. The VGG16 is trained in color images of tensor shape 224×224×3.

Examples of the application of other 3×3 kernels are shown in Figure 2.5.



$$\begin{bmatrix} 0.3 & 0 & 0 \\ 0.3 & 0 & 0 \\ 0.3 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.3 \end{bmatrix} \qquad \begin{bmatrix} 0 & -0.1 & 0 \\ -0.1 & 0.5 & -0.1 \\ 0 & -0.1 & 0 \end{bmatrix}$$

Figure 2.5: Examples of other 3×3 kernels applied to the same 8×8 original image.

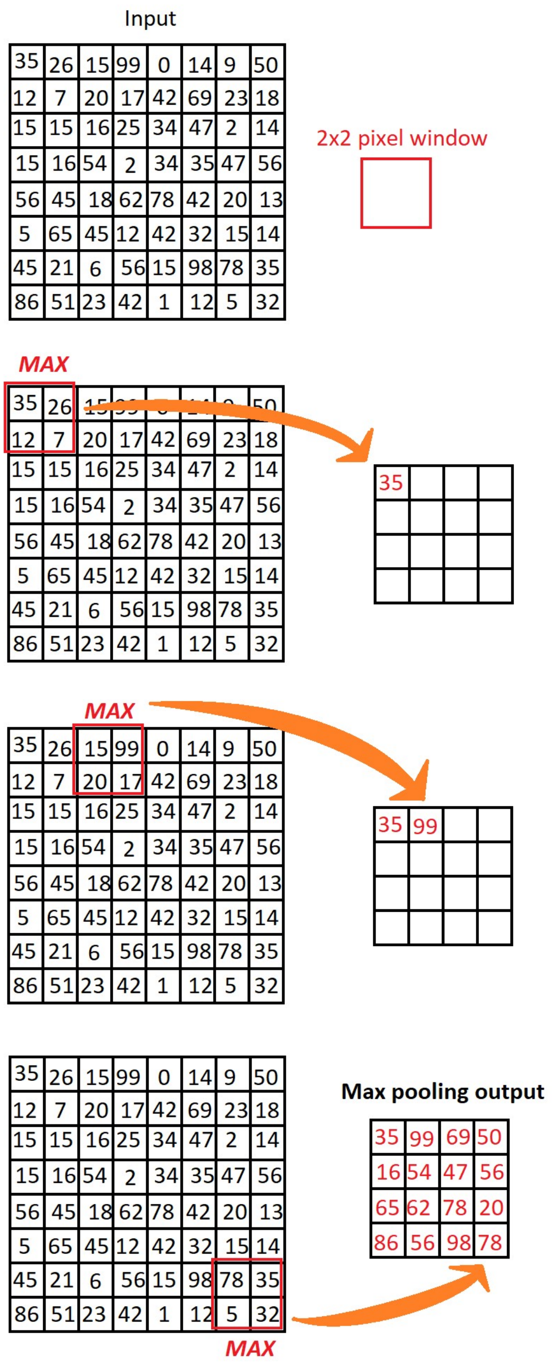
Convolutions can be used to, among other things, sharpen colors, detect edges, and blur the image.

2. Pooling

A pooling layer is an important building block of a neural network. It reduces the number of parameters in the network, in order to simplify computation. Without it, research in the AI fields would be considerably harder.

The pooling operation consists of, given a pixel window, choosing one pixel as an output. Similarly to the convolution process, this pixel window slides over all pixels. The pooling operation used in the VGG16 architecture is the “max pooling”.

The max pooling operation occurs by choosing the largest pixel value within the pixel window. In the VGG16, a 2×2 pixel window is used. This results in an output 1/4 the input size, as illustrated in Figure 2.6.



35	26	15	99	0	14	9	50
12	7	20	17	42	69	23	18
15	15	16	25	34	47	2	14
15	16	54	2	34	35	47	56
56	45	18	62	78	42	20	13
5	65	45	12	42	32	15	14
45	21	6	56	15	98	78	35
86	51	23	42	1	12	5	32

35	99	69	50
16	54	47	56
65	62	78	20
86	56	98	78

Figure 2.6: The VGG16 architecture uses a 2×2 max pooling with stride 2. It reduces the output to 1/4 the input size.

Some other pooling operation are: the average pooling (chooses the average number in the window); the global max pooling (chooses the largest number in the entire input matrix. This transforms the entire matrix into a single number); and the global average pooling (chooses the mean of the entire input matrix. This transforms the entire matrix into a single number).

The overall architecture of the VGG16 is shown in Figure 2.7.

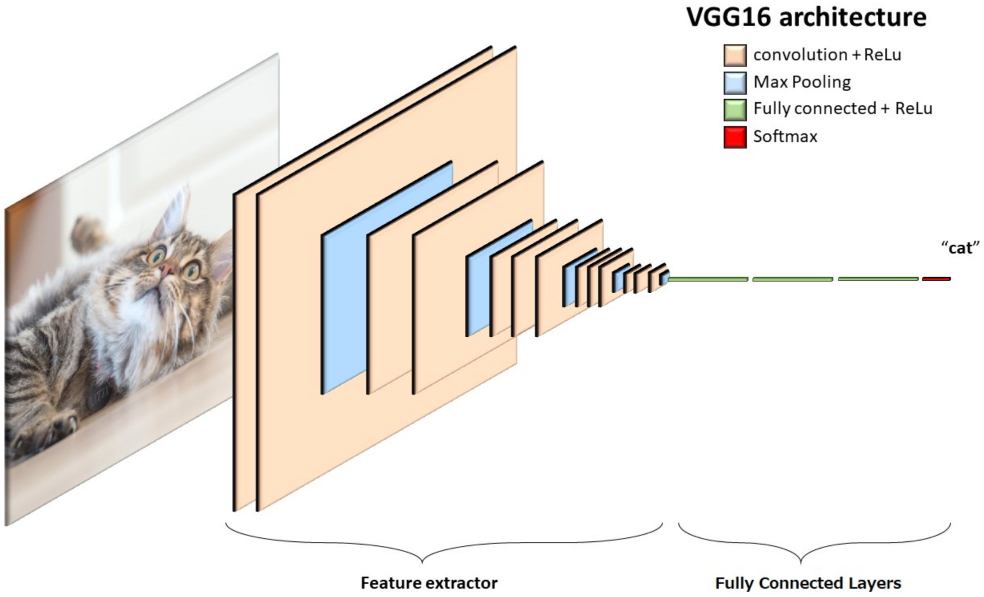


Figure 2.7: The VGG16 structure is composed of two parts: the convolution part, also called feature extractor; and the fully connected part, also called classifier (see Section 2.2.5).

2.2.2 ResNet50

The ResNet50 is a network proposed by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun from Microsoft Research. It took part on the "ILSVRC and COCO 2015" competitions, where it won first place on the ImageNet detection, ImageNet localization, COCO detection and COCO segmentation [14].

ResNet stands for "residual network". A residual network has, in addition to its convolutional/pooling consecutive layers, direct information paths between its building blocks, as depicted in Figure 2.8. In this way, the basic building block initially implement identity maps, and they are able to learn the difference between those identity maps and the operations that they must perform. In the original paper "Deep Residual Learning for Image Recognition" [14], it is hypothesised that these direct information paths can make the learning process easier in the case of functions closer to identity maps. It also mitigates the problem of gradient information degradation throughout the stacked layers. Training the residual network is called "residual learning".

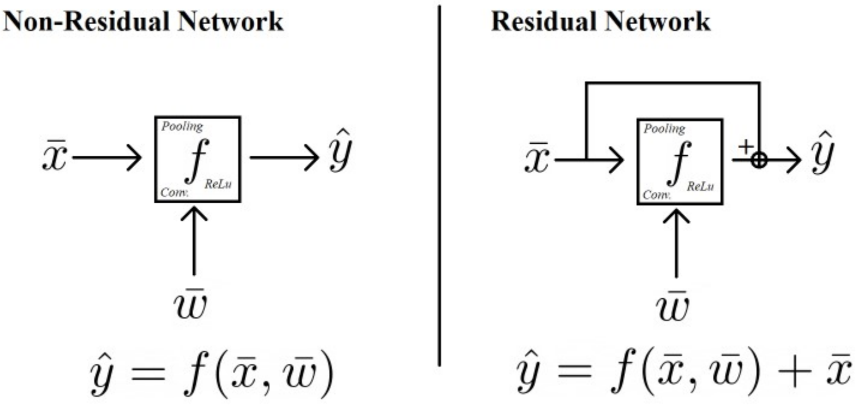


Figure 2.8: In a residual network, output $\hat{y} = f(\bar{x}, \bar{w}) + \bar{x}$. It is hypothesized that training $f(\bar{x}, \bar{w}) = \hat{y} - \bar{x}$ is an easier task than training $f(\bar{x}, \bar{w}) = \hat{y}$.

The ResNet50 architecture follows the same principle as the "34-layer residual" architecture described in the original paper [14], but its depth is extended to 50 layers.

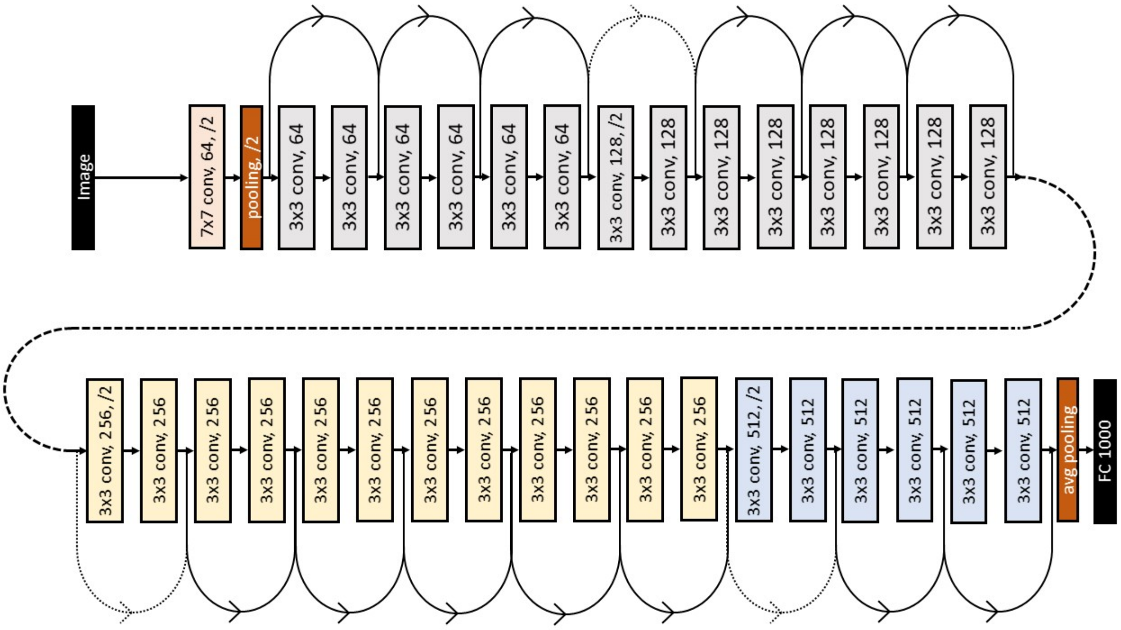


Figure 2.9: ResNet34 architecture. The ResNet50 network follows the same principle as the ResNet34, but its depth is extended to 50 layers.

2.2.3 Inception-V3

Since the appearance of AlexNet [15] in the ImageNet contest 2012, convolutional neural networks have become better at correctly classifying images. VGG16 as well as other models have shown great results in terms of classification accuracy. These models however have become larger and very computationally expensive.

Aiming at mobile market possibilities, the Inception-V3 is a network proposed by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens and Zbigniew Wojna, from Google. It was made to be computationally efficient, while not compromising its performance. Using a reduced number of parameters, factorized convolutions and some aggressive regularization methods, it introduced substantial gains over the previous state-of-the-art networks [16].

With this objective, the Inception-V3 applies the following techniques:

- Factorizing Operations into Smaller Convolutions

  Inception-V3 focuses on having smaller convolution kernel sizes rather than larger ones. This reduces the number of trainable parameters and leads to faster training. This is done by factorizing large kernels into stacked smaller ones. For example, one 5×5 convolution kernel is replaced by two 3×3 convolutions, thus reducing the respective number of trainable parameters from 25 to 18. Factorized convolutions have already been shown to outperform traditional convolutions on performance/complexity ratio [17].

- Asymmetric Convolutions

  Replacing a full 3×3 convolution by a combination of 3×1 and 1×3 convolution accelerates network evaluation considerably. This has little impact on the model performance [18]. This way, a 3×3 asymmetric convolution has slightly fewer parameters than two stacked 2×2 convolutions, while performing better. In fact, any $n \times n$ convolution can be made with a $1 \times n$ and $n \times 1$ operations, but this have shown to affect the models performance on early convolution layers, thus being used only at later stages.

- Auxiliary Classifier

Auxiliary classifiers were introduced in the first Inception paper as a method for improving training convergence in very deep networks. They were therefore initially used in the Inception-V3 with the same objective, but ended up acting as a regularization method.

- Grid Size Reduction

Grid size reduction is usually done by a convolution followed by a pooling operation. In this way, the cost of processing is predominantly the cost of the convolution. The Inception-V3 proposes a new, more efficient way of reducing the number of parameters, by mixing smaller pooling and convolution operations into one layer.

The Inception-V3 overall architecture is shown in Figure 2.10.

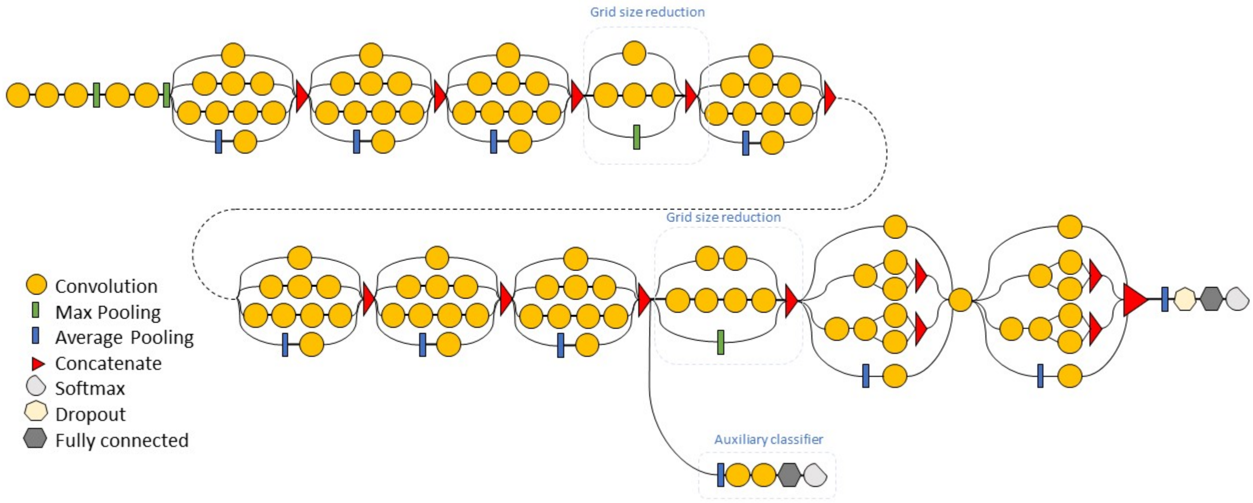


Figure 2.10: Inception-V3 overall architecture.

2.2.4 Long Short-term Memory

A basic neural network typically maps an input obtained at a specific time into an output referring to that same time instant. It traditionally does not use data from previous time instants in order to generate its current output, *i.e.* its output referring to the present time instant. Some problems however are intrinsically time dependent. For example, when trying to translate a sentence, the neural network must consider previous words in order to predict the next one, as illustrated in Figure 2.11.

I love you
Eu te amo
好きです

Figure 2.11: Translation is a heavily time and context dependent process. Translating word by word is not an option.

Recurrent neural networks are a type of neural network especially designed to process time dependence. Their main characteristic lies in having an internal loop that allows previous information to be represented and to remain available for computation (Figure 2.12).

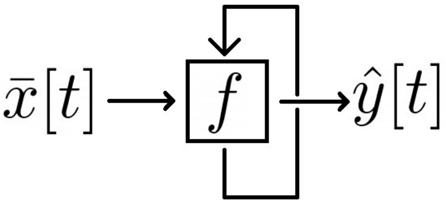


Figure 2.12: RNNs have an internal loop that allows information to be represented and to remain available for computation. This way they can be used to process temporal information.

Figure 2.13 shows the loop unrolling and how time is handled more clearly.

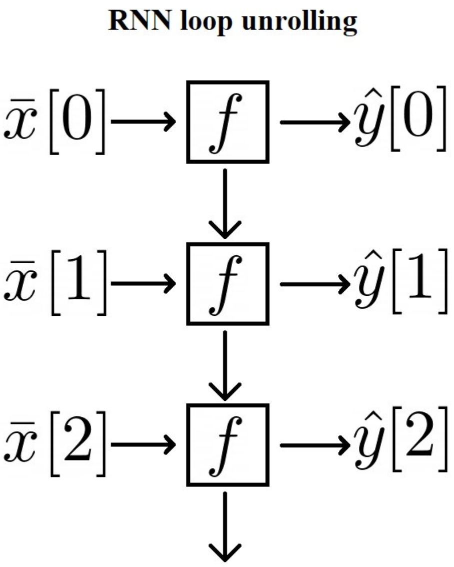


Figure 2.13: RNNs loop unrolled for each time step. It is clear that information at $t = 0$ affects output at $t = 2$.

Classical RNNs have trouble retaining information for long sequences [19]. This can be a problem if the output depends on information from the distant past. The long short-term memory (short LSTM) is a recurrent neural network cell proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [20]. It addresses the problem of learning from long sequences, having no problem in retaining information. It is widely used today as a building block in neural networks that handle temporal structures within the data.

The internal structure of an LSTM is not composed by a unique weight layer, but by four (Figure 2.14). Each layer plays a special role on the LSTM processing. Mathematically, the LSTM structure is described as:

$$\bar{f}[t] = sig(W_f\bar{x}[t] + U_f\bar{h}[t-1] + \bar{b}_f) \tag{2.13}$$

$$\bar{i}[t] = sig(W_i\bar{x}[t] + U_i\bar{h}[t-1] + \bar{b}_i) \tag{2.14}$$

$$\bar{o}[t] = sig(W_o\bar{x}[t] + U_o\bar{h}[t-1] + \bar{b}_o) \tag{2.15}$$

$$\bar{c}_*[t] = \tanh(W_c\bar{x}[t] + U_c\bar{h}[t-1] + \bar{b}_c) \tag{2.16}$$

$$\bar{c}[t] = \bar{f}[t] \odot \bar{c}[t-1] + \bar{i}_t \odot \bar{c}_*[t] \tag{2.17}$$

$$\bar{h}[t] = \bar{o}[t] \odot \tanh(\bar{c}[t]) \tag{2.18}$$

where:

$\bar{x} \in \mathbb{R}^d$	input vector of LSTM
$\bar{f} \in \mathbb{R}^u$	forget gate activation vector
$\bar{i} \in \mathbb{R}^u$	input gate activation vector
$\bar{o} \in \mathbb{R}^u$	output gate activation vector
$\bar{h} \in \mathbb{R}^u$	hidden state output vector
$\bar{c}_* \in \mathbb{R}^u$	cell input activation vector
$\bar{c} \in \mathbb{R}^u$	cell activation vector
$W \in \mathbb{R}^{u \times d}$	input weight matrix
$U \in \mathbb{R}^{u \times u}$	hidden state weight matrix
$\bar{b} \in \mathbb{R}^u$	bias vector
d	input vector size
u	number of LSTM hidden units
sig	sigmoid activation function
tanh	hyperbolic tangent activation function

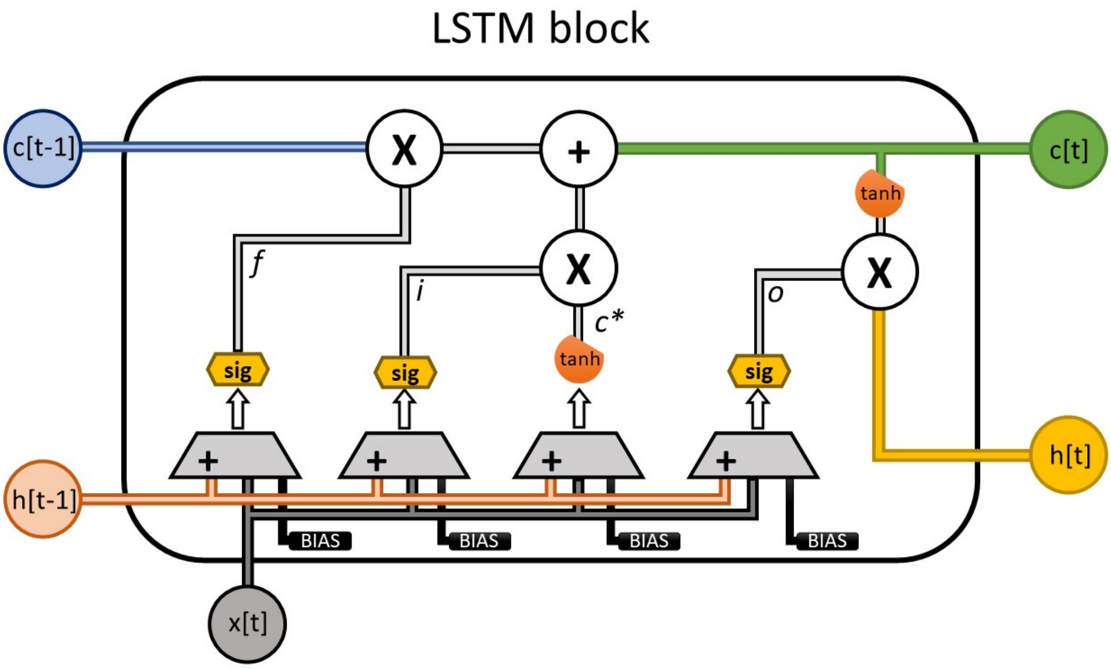


Figure 2.14: LSTM cell structure.

Looking at the LSTM as a black box:

- LSTM input is the vector $\bar{x}[t]$.
- LSTM output is the vector $\bar{h}[t]$.
- State information for the next time instant is passed through the output vector $\bar{h}[t]$ and through the cell activation vector $\bar{c}[t]$.

2.2.5 Transfer Learning

A model optimized on a task can often be used in another task, if the dataset for the new task is related to the original one. This is called transfer learning, and it corresponds to the idea that an already optimized model can be applied to other tasks with little change. Usually, the original task is optimized over a big dataset, and the model for the new task can benefit from the previous model generalization.

In the case of image processing with CNNs, basic models can be divided into two parts:

- Feature Extractor

This is the part where convolution and pooling layers are present. As the name suggests, it extract features from the images. Features can be understood as a denser, lower-dimensional space representation of the image original information.

- Fully-connected Layers (FC)

The FC layers (two of them are shown in Figure 2.15) are where the image information is mainly classified or regressed. They are the last layers (*i.e.* the layers closest to the output) in the models presented in Sections 2.2.1, 2.2.2 and 2.2.3.

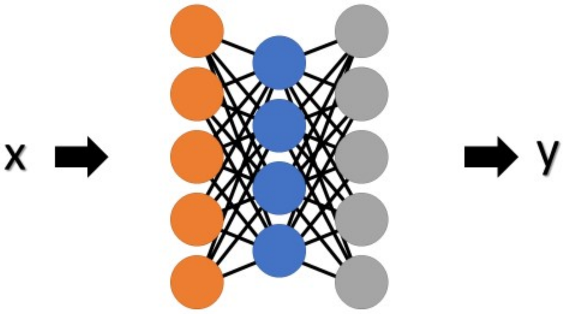


Figure 2.15: Example of FC layer structure.

On CNN based models, it is usual to transfer the feature extractor part to the new model, and train only the last FC layers on the new task. This way, it is possible to save time and resources not training the feature extractor, and benefit from the broader dataset it was trained upon. It also expands the possibilities of research on lower hardware, as training the feature extractor demands a considerably amount of memory and GPU power.

2.3 Data Processing

Until now, a brief review about machine learning, some basic training algorithms, some commonly used convolutional and recurrent neural networks, and the concept of transfer learning were discussed. In this section, a brief review of data processing is discussed. Processing is done before and after the training process.

2.3.1 Pre-Processing: Data Normalization

It is important to normalize the data before training takes place. Normalized data are handled by the network more easily, and they yield better training outcomes. There are some different forms of data normalization. The data standardization (Z-score normalization) was used in this research. For a random dataset, one can usually assume it to be drawn from a probability density function that is approximately Gaussian:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \tag{2.19}$$

where μ is the mean and σ is the standard deviation of the probability density function. Normalizing and standardizing a Gaussian distribution function means setting $\mu = 0$ and $\sigma = 1$. This process is done by subtracting the dataset mean from all data arrays, and then dividing the resulting arrays by the dataset standard deviation, as illustrated in Figure 2.16.

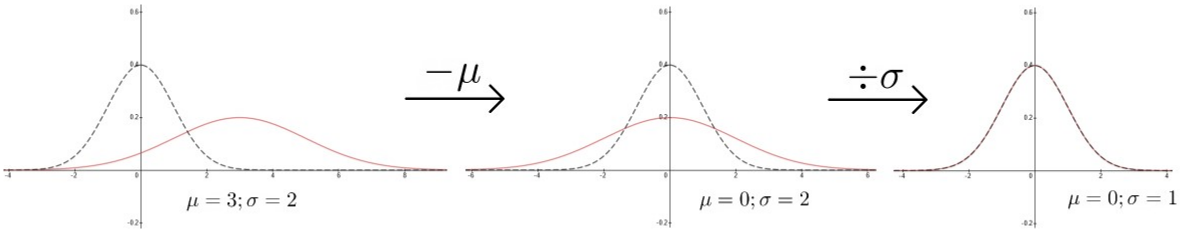


Figure 2.16: Process of Gaussian normalization.

Dataset mean and standard deviation are respectively calculated as:

$$\mu_{\bar{x}} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{2.20}$$

$$\sigma_{\bar{x}} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N-1}} \tag{2.21}$$

Normalizing the dataset is specially important when using a transfer learning approach. Because weights that were optimized for another dataset are used, having the dataset mean and standard deviation equal to those of the optimized dataset usually yields better results.

In the case of 3-channel image datasets (dataset with color images), normalization is usually independent for each color channel.

2.3.2 Post-Processing: Moving Average and Correlation Coefficient

When evaluating regression results, it is interesting to filter the neural network outputs in order to compute a temporal average of the output sequence. A simple filter that can be used for this is the moving average filter. The moving average filter outputs the average of the T previous outputs:

$$y[n] = \frac{1}{T}\sum_{k=0}^{T-1} x[n-k]. \tag{2.22}$$

For example, for $T = 4$:

$$y[n] = \frac{1}{4}(x[n] + x[n-1] + x[n-2] + x[n-3]). \tag{2.23}$$

The Z transform of such filter is:

$$Y(z) = \frac{1}{4}(1 + z^{-1} + z^{-2} + z^{-3})X(z). \tag{2.24}$$

Or more generically:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{L}\sum_{k=0}^{L-1} z^{-k}. \tag{2.25}$$

By setting $|z| = 1$ and substituting $z^{-k} = e^{-j\omega k}$, we can see that the moving average is a low-pass filter. For example, using $T = 2$:

$$H(e^{j\omega}) = \frac{1}{2}(1 + e^{-j\omega}). \tag{2.26}$$

Applying Euler's formula for $e^{j\theta} = \cos\theta + j\sin\theta$ and $|z| = \sqrt{\Re(z)^2 + \Im(z)^2}$, the modulus of function $H(z)$ is found, as seen in Equation 2.27 and Figure 2.17.

$$|H(e^{j\omega})| = \frac{1}{2}\sqrt{(1 + \cos(\omega))^2 + \sin(\omega)^2}. \tag{2.27}$$

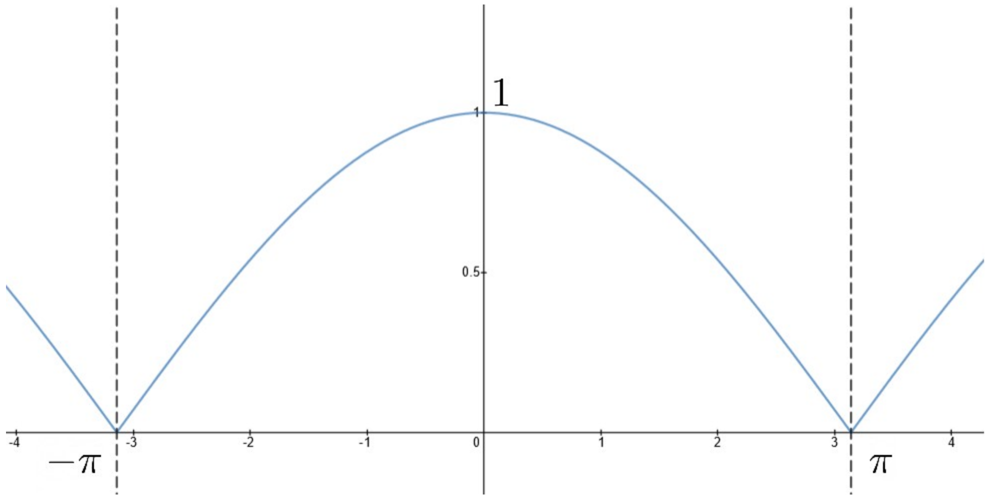


Figure 2.17: Moving average frequency response for $L = 2$. Graph of $\omega \to |H(e^{j\omega})|$. For the maximum frequency $\omega = \pi$, $|H(e^{j\omega})| = 0$.

The moving average is used as a low-pass filter for data smoothing.

For comparing how much two random variables (the average output value and the target sound pressure, in our case) are related, the Pearson correlation coefficient (from now on, correlation coefficient) is used. The correlation coefficient ρ ranges from -1 to 1. Correlation coefficient equals to 0 means the two variables are totally uncorrelated (there is no relationship between then). Correlation coefficient equal to 1 means they are totally correlated, and -1 totally inversely correlated.

Correlation coefficient between random variables $\bar{X}$ and $\bar{Y}$ is given by:

$$\rho_{\bar{X}\bar{Y}} = \frac{Cov(\bar{X}, \bar{Y})}{\sigma_{\bar{X}}\sigma_{\bar{Y}}}. \tag{2.28}$$

Where $Cov(\bar{X}, \bar{Y})$ is the covariance between the random variables $\bar{X}$ and $\bar{Y}$:

$$Cov(\bar{X}, \bar{Y}) = \frac{1}{N}\sum_{i=0}^{N}(x_i - \mu_{\bar{X}})(y_i - \mu_{\bar{Y}}). \tag{2.29}$$

The mean squared error (MSE) is defined as

$$\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2, \tag{2.30}$$

where N is the length of the sequence; Y_i is the target output; and $\hat{y}_i$ is the predicted output.

Chapter 3

Methodology

In this chapter, the methodology used in this work is presented. This includes the dataset that is used, how it is processed, and the networks that are developed and evaluated on this work.

All code in this research was executed on “Febe”, which is the deep learning computer server from the Digital and Analog Signal Processing Laboratory (PADS). It is composed of a 6 core/12 thread Intel® Core™i7-6850K CPU @ 3.60 GHz, 64 GB of RAM, and four 28 core, 10.92 GB GeForce GTX 1080 Ti @ 1.582 GHz GPUs. We use TensorFlow 2.1.0 back-end with Keras and CUDA 10.1.

3.1 Structure and Implementation

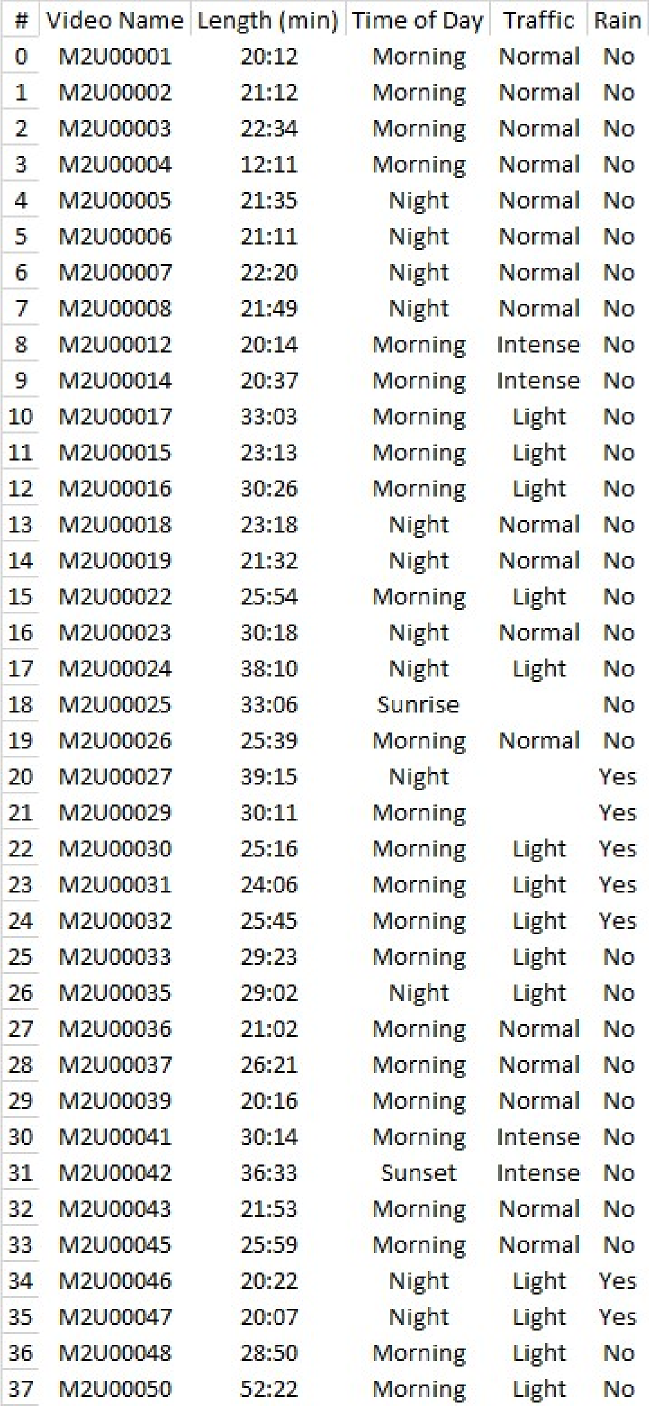
In this first section, the methodology proposed in this research is described.

3.1.1 Dataset Generation

The problem in this research consists of predicting the output sound pressure level of a traffic scene, using only visual information (a single image or an image sequence) as an input. In this application, traffic scenes are restricted to street videos captured from a single point of view.

To train deep-learning models for sound pressure prediction using only video frames as input, a dataset composed by 38 video files with an overall length of 995 minutes was used. All videos were recorded from the same place and angle, approximately 50 meters away from the are where the traffic noise is supposed to be predicted, in different week days, at different morning, afternoon, and night times, and under different weather and illumination conditions. This dataset is an expanded version of the one used on [9], in which it was used 10 videos with overall length of 203 minutes. Detailed list of videos can be seen in Table 3.1.

Table 3.1: Detailed list of videos in the available dataset.



#	Video Name	Length (min)	Time of Day	Traffic	Rain
0	M2U00001	20:12	Morning	Normal	No
1	M2U00002	21:12	Morning	Normal	No
2	M2U00003	22:34	Morning	Normal	No
3	M2U00004	12:11	Morning	Normal	No
4	M2U00005	21:35	Night	Normal	No
5	M2U00006	21:11	Night	Normal	No
6	M2U00007	22:20	Night	Normal	No
7	M2U00008	21:49	Night	Normal	No
8	M2U00012	20:14	Morning	Intense	No
9	M2U00014	20:37	Morning	Intense	No
10	M2U00017	33:03	Morning	Light	No
11	M2U00015	23:13	Morning	Light	No
12	M2U00016	30:26	Morning	Light	No
13	M2U00018	23:18	Night	Normal	No
14	M2U00019	21:32	Night	Normal	No
15	M2U00022	25:54	Morning	Light	No
16	M2U00023	30:18	Night	Normal	No
17	M2U00024	38:10	Night	Light	No
18	M2U00025	33:06	Sunrise		No
19	M2U00026	25:39	Morning	Normal	No
20	M2U00027	39:15	Night		Yes
21	M2U00029	30:11	Morning		Yes
22	M2U00030	25:16	Morning	Light	Yes
23	M2U00031	24:06	Morning	Light	Yes
24	M2U00032	25:45	Morning	Light	Yes
25	M2U00033	29:23	Morning	Light	No
26	M2U00035	29:02	Night	Light	No
27	M2U00036	21:02	Morning	Normal	No
28	M2U00037	26:21	Morning	Normal	No
29	M2U00039	20:16	Morning	Normal	No
30	M2U00041	30:14	Morning	Intense	No
31	M2U00042	36:33	Sunset	Intense	No
32	M2U00043	21:53	Morning	Normal	No
33	M2U00045	25:59	Morning	Normal	No
34	M2U00046	20:22	Night	Light	Yes
35	M2U00047	20:07	Night	Light	Yes
36	M2U00048	28:50	Morning	Light	No
37	M2U00050	52:22	Morning	Light	No

From each video, both audio and visual information are extracted. Audio information is extracted at 48000 samples/second.

Visual information consists of frames that were extracted from all videos. The frame extracting process consisted of resizing the videos, and down-sampling the output frames. Video resizing was initially done using the ffmpeg program, as described in [9]. This program however introduced compression artifacts that degraded the final image quality. Resizing and images extraction was then conducted using the python library OpenCV. Frames were resized from "720×480" to "224×224" pixels. All frames in each video were extracted and downsampled by a factor of 1/10. This way, the time interval between subsequent extracted frames is approximately 1/3 of a second (3 frames = 1 second). Overall extracting process is depicted in Figure 3.1.

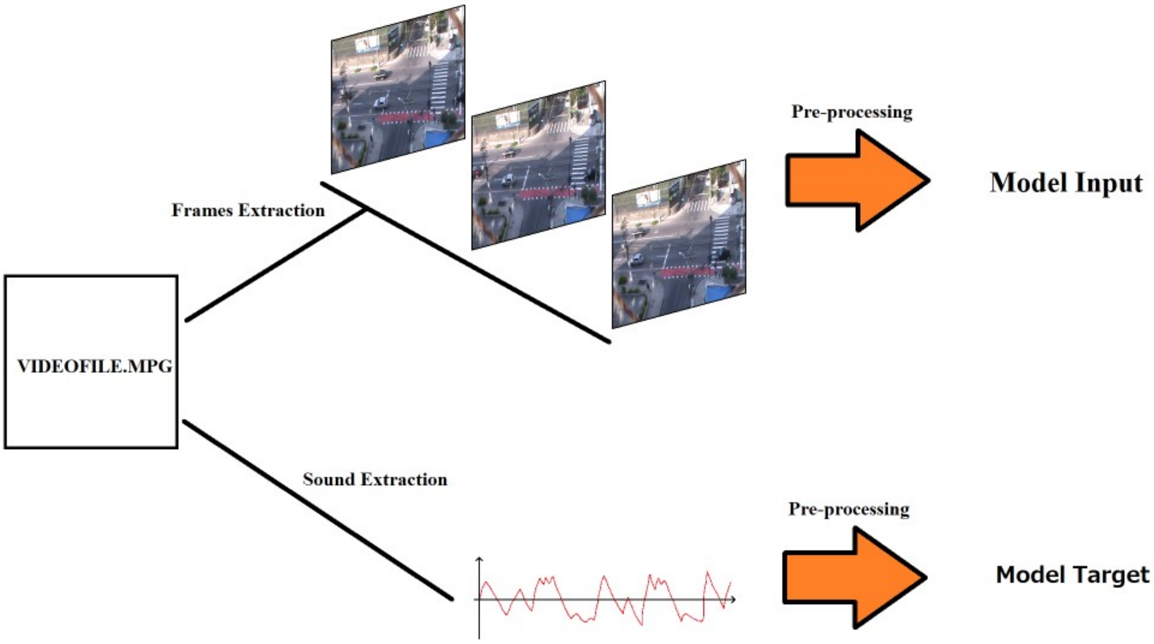


Figure 3.1: Video extraction process.

3.1.2 Data Pre-Processing

Input data pre-processing consists of normalizing the extracted video frames using the mean and standard deviation, as described in Section 2.3.1. Normalization has to be applied independently for different folds, because they have different combinations of videos. For that reason, the input data is normalized only when building each fold (details are given in Section 3.1.3). Normalization is applied independently for each color channel.

After extracting the audio from video, the sound pressure is calculated as:

$$S_t[i] = \ln\left(\frac{1}{M}\sum_{k=iM}^{(i+1)M} I^2[k]\right) \text{,where } S_t \text{ is the sound pressure,} \tag{3.1}$$

$$M = \frac{\text{number of audio samples on video}}{\text{number of extracted frames on video}}, \text{ and} \tag{3.2}$$

$$I[k] = \text{Audio sample at k} \tag{3.3}$$

The expression in Equation 3.1 is non-causal. For the computation of the present sound pressure sample, it requires storage of the future M audio samples, which does not create any problem for offline training. For online training or online evaluation, non-causality would lead to a minor delay (around half the 1/3 second between successive frames) in the neural network output evaluation instant, with respect to the present time instant. This calculation, illustrated in Figure 3.2, is done using the scientific python library numpy, specially optimized for matrix operations.

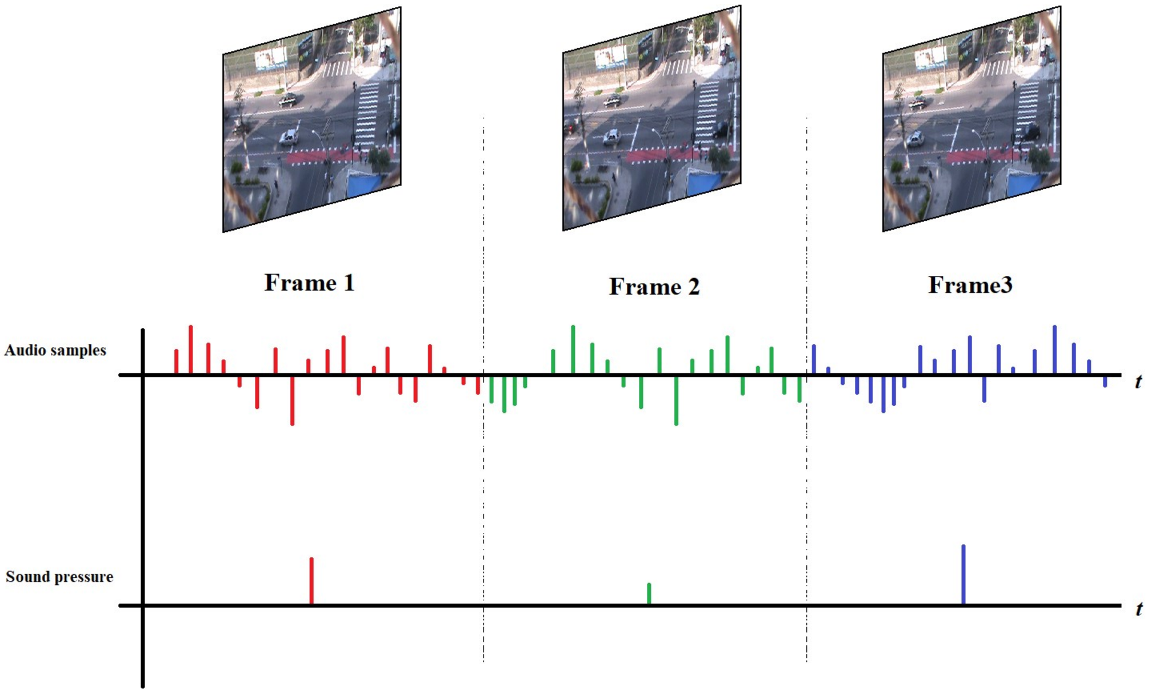


Figure 3.2: Sound pressure calculation.

The last output pre-processing stage consists of transforming the stereo audio file into a mono audio file, if necessary, by adding both channels and dividing the result by two. Normalizing the sound pressure (target) sequences for zero mean and unitary standard deviation did not improve training results, so the target sequences are used without any normalization.

3.1.3 Fold Generation

Fold generation process consists of using parts of the available dataset to create smaller datasets with different video combinations for training and test. The 38 videos were divided into 10 different folds, as seen in Appendix A.

Folds are diversified by selecting the videos based on their characteristics, specially daytime, while trying to maintain all folds with approximately 25% of test data, based on the length of each video. This corresponds to roughly 11 to 12 hours of training data and 3.5 to 4.5 hours of test data. Videos number 30 and 31 present the model with an adverse situation, with an accident at the intersection, causing an uncommon traffic behaviour. Those two videos were included only in the two last folds.

Normalization of input images occurs when each the data files for each fold are created. For this, calculation of the individual mean and standard deviation values for each video in the dataset separately is first executed. This can be done using the scientific python library numpy. With these individual values, the mean and standard deviation values for the fold can be calculated using the previously calculated values, as

$$\mu_{fold} = \frac{\sum_{videos}(\mu_{video} \times videoLength)}{\sum_{videos} videoLength}, \tag{3.4}$$

$$\sigma_{fold} = \sqrt{\frac{(t_{xx} - t_x^2)/t_n}{t_n - 1}}, \text{ where} \tag{3.5}$$

$$t_n = \sum_{videos} videoLength, \tag{3.6}$$

$$t_x = \sum_{videos} (\mu_{video} \times videoLength), \text{ and} \tag{3.7}$$

$$t_{xx} = \sum_{videos} \left(\sigma_{video}^2 (videoLength - 1) + \frac{(\mu_{video} \times videoLength)^2}{videoLength} \right) \tag{3.8}$$

and normalize the fold with those values. This procedure is useful for saving hardware resources, as each video can be processed separately. Calculating mean and standard deviation using all videos at once requires an unnecessary amount of memory (in this case, about 108 GB of memory). This way, μ and σ can be calculated for each video separately, and the fold statistics calculated later on.

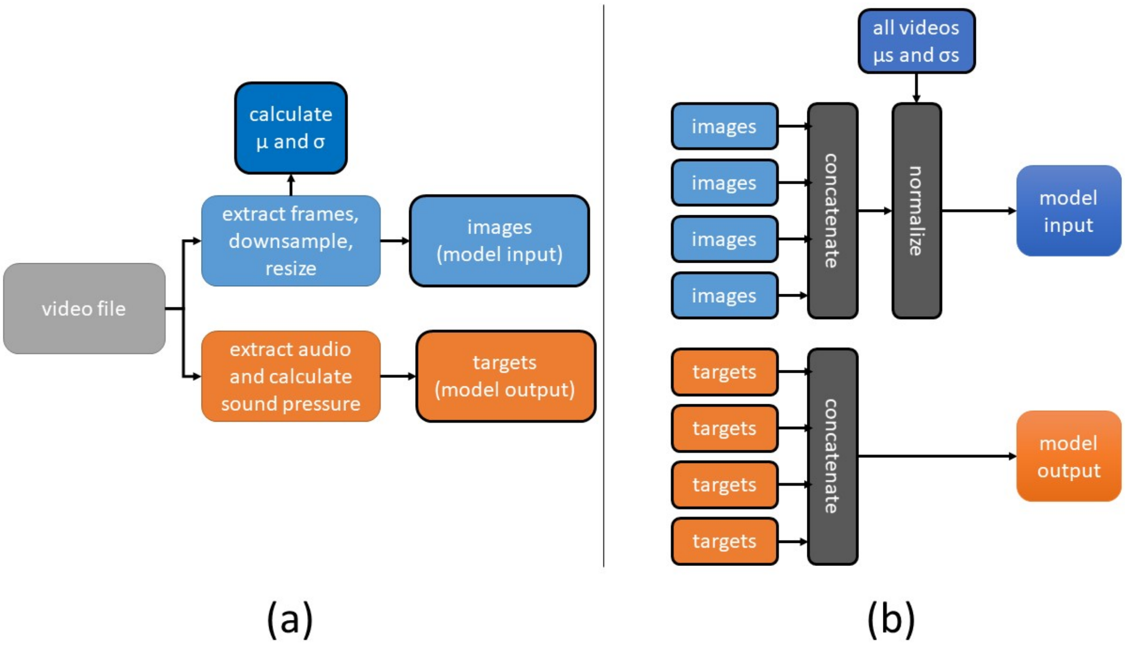


Figure 3.3: Data generation: (a) how a single video is processed. In this stage, three files are generated, containing the images, targets and the videos mean and standard deviation; (b) how the models input and output files are generated. All video images are concatenated and normalized into a single file. Targets are also concatenated, but are not normalized.

This process in Figure 3.3 generates two files for each fold: one for input data and one for output data. Input file consists of a tensor of shape: "number of frames"×224×224×3. The first index of this tensor refers to the frames of the dataset. Output file consists of a vector of shape number of frames×1. Each position of the frame at the same ("number of frames") time index.

The final generated fold files are considerably larger than the original video files, because when normalizing input frames, files are converted into a 32-bit float representation. This causes each fold input file to have size up to 100 GB. Files of this size are hardly manageable on the computer (see page 27), and they would rapidly occupy all free disk space. Files of this size also lead to slow training, requiring much more computer power than there is usually available.

To handle this problem, transfer learning concepts presented in Section 2.2.5

are applied. Using the downloaded VGG16, ResNet50 and Inception-V3 models pre-trained on the ImageNet dataset, features from all the input data were extracted before training. After a final pooling layer, this reduces the size of the training and test sets to 1.3% (ResNet50, InceptionV3) and 0.3% (VGG16) of their original size.

3.2 Models

In this section, the models featured in this research are described.

3.2.1 Base Models

Base models are divided in LSTM based architectures and non-LSTM based architectures. Both models use transfer learning, with features extractors trained upon the ImageNet dataset.

3.2.2 Non-LSTM based models

The non-LSTM based models are the simplest. Two architectures are tested, with and without a non-linear hidden layer. Basic models are shown in Figure 3.4.

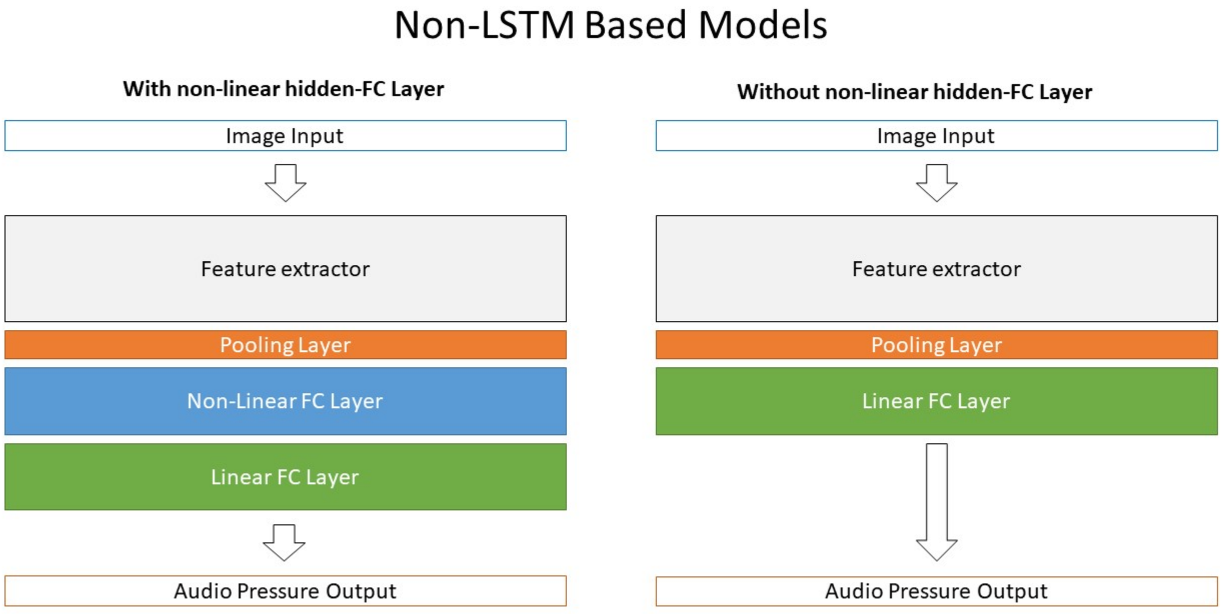


Figure 3.4: Non-LSTM based models.

In regards to each layer, we considered the following different configurations:

- Feature extractor: VGG16, ResNet50 and InceptionV3.
- Pooling layer: global max pooling (GMP) and global average pooling (GAP).
- Non-linear FC layer: activation (relu or tanh) and regularization (L2 or none).

3.2.3 LSTM Based Models

LSTM based models are a modification of the non-LSTM, based on the work done by Carreira, Zisserman [21] on LSTM for image networks. Four variations of LSTM network are tested, with and without a non-linear hidden layer, and with LSTM models before or after the FC layers. Basic architectures are shown in Figure 3.5.

Dataset loading to the LSTM requires some special attention. A frame window of length N is used to load the dataset into the model. This window is slided frame by frame, and consider either the last frames output sound pressure (called causal prediction) or the middle's output sound pressure (called non-causal prediction), as seen on Figure 3.6. When sliding this window, one must not use frames from different videos on the same window.

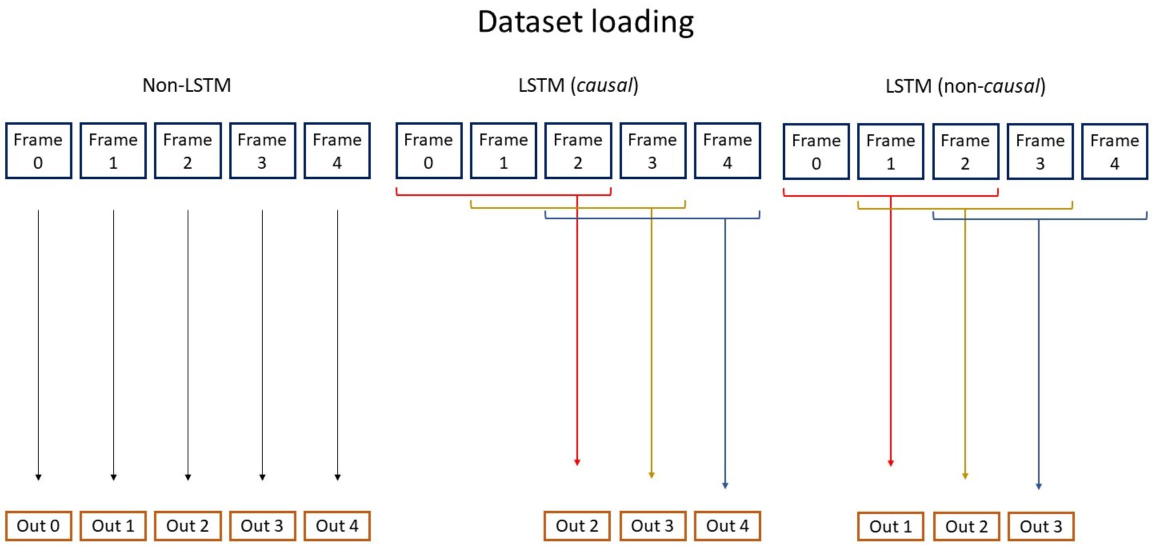


Figure 3.6: Dataset loading.

In regards to each layer and dataset loading, the following different configurations are considered:

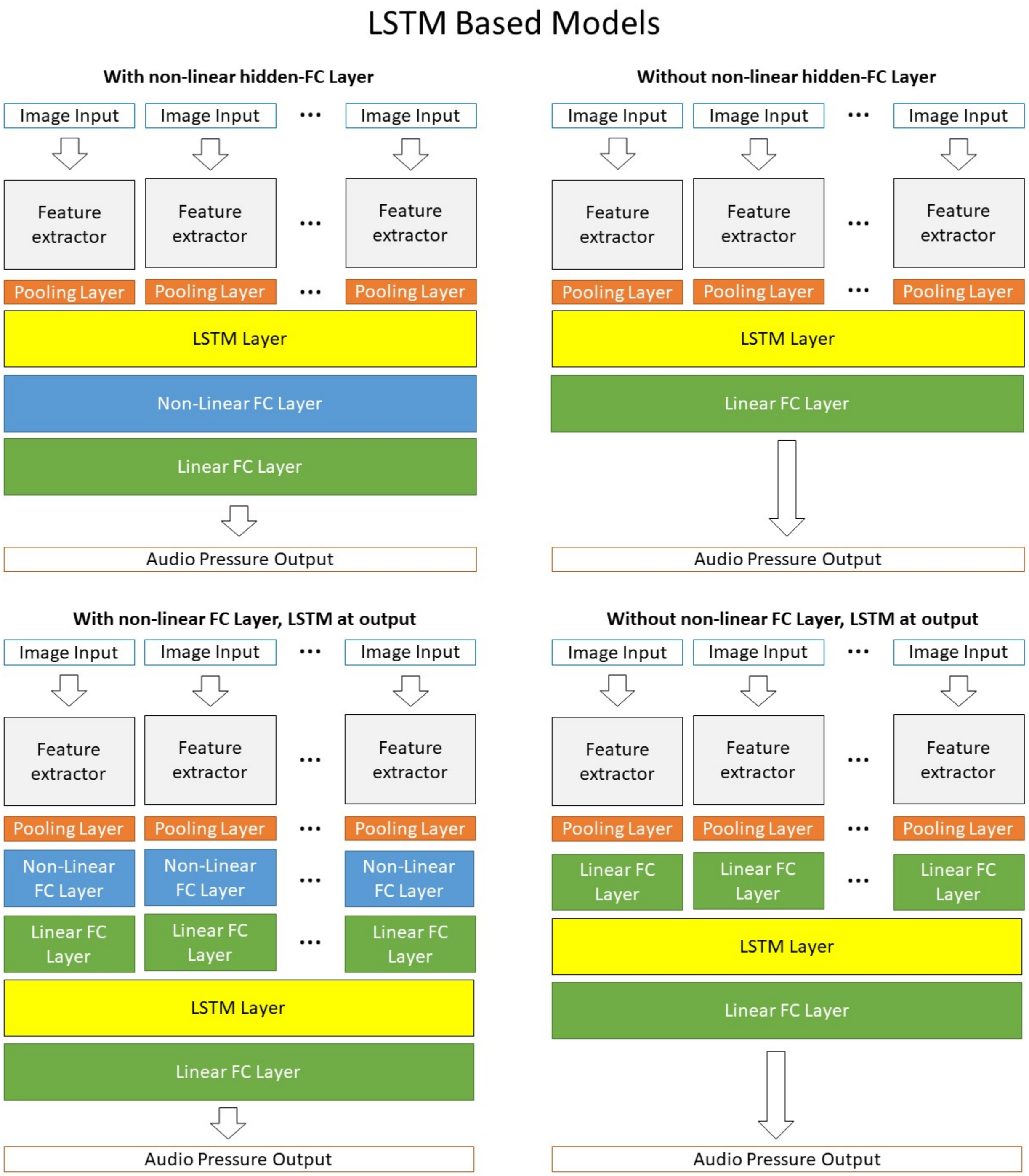


Figure 3.5: LSTM based models.

- Feature extractor: VGG16, ResNet50 and InceptionV3.
- Pooling layer: global max pooling (GMP) and global average pooling (GAP).
- Non-linear FC layer: activation (relu or tanh) and regularization (L2 or none).
- LSTM layer: dropout (0% or 20%), number of frames (9, 17 or 32) and statefulness (stateful or stateless).
- Dataset loading: different window sizes and causality of output.

Despite the overwhelming number of hyperparameters and possible combinations, not all possible configurations are tested, as it would require a considerable and unnecessary amount of time and resources (over 6 months of processing on Febe (see Page 27)). Instead, a smaller test was conducted in order to choose which hyperparameters were the most relevant to the research (refer to Section 4.1).

3.2.4 Models with Auxiliary Inputs

As a last model, an experiment using a simple auxiliary input on the base model was conducted, similarly to the optical flow input shown in [21]. Faster R-CNN extracted features from each frame were used as an auxiliary input. The extractor was trained over the Microsoft COCO dataset [22]. Features include a list of identified objects, object class, confidence score and bounding boxes coordinates. Models architecture can be seen in Figure 3.7.

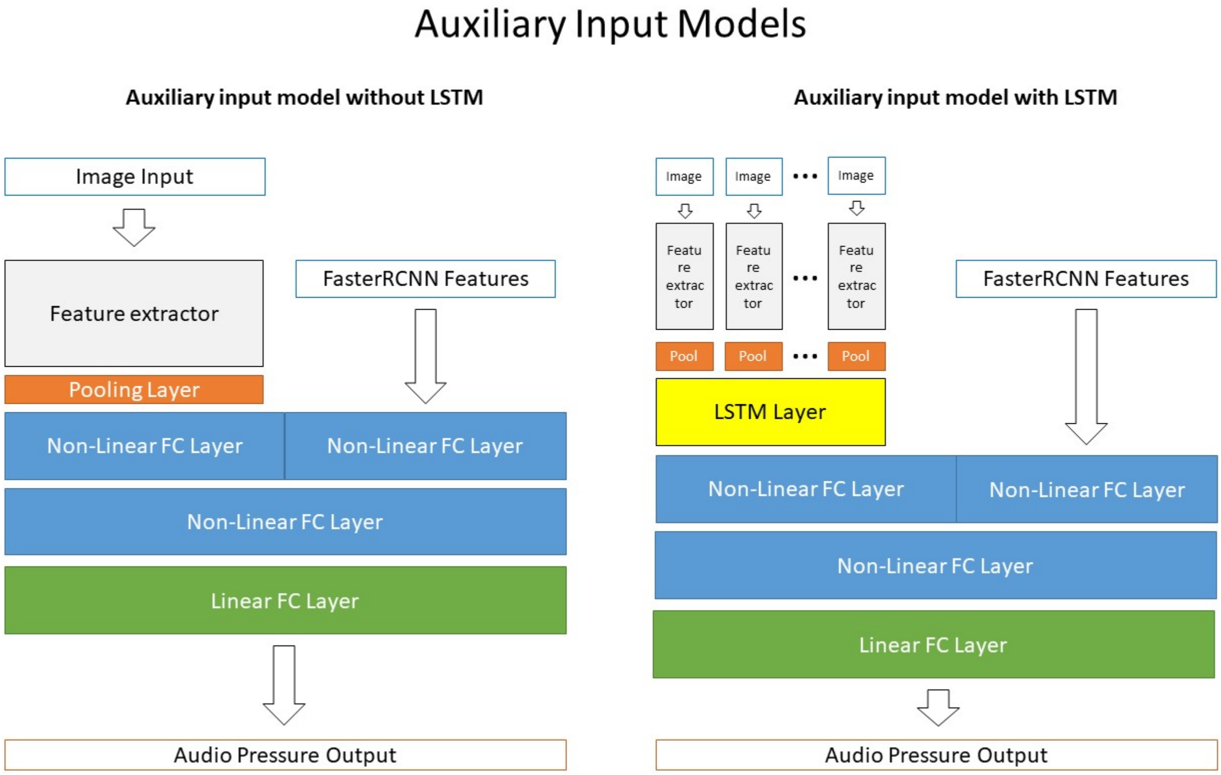


Figure 3.7: Auxiliary input models.

Processing of these features consisted of ignoring identified objects with confidence score bellow 70%, and excluding objects considered irrelevant and/or misidentifications. Example of misidentified objects are “fork”, “tv” and “wine glass”. After this, only 6 categories were deemed relevant: “car”, “person”, “bicycle”, “motorcycle”, “truck” and “bus”.

Two different tensor representations of this information were considered, as seen in Figure 3.8.

First representation consider a mixture of all information extracted from the Faster R-CNN network. This representation was called as “one-hot representation”. Each image has a matrix of features. Each line contains a feature with score, area of the bounding box and a one-hot representation of the class. It was decided that each image can have at most 20 different features.

Second and simpler representation consists of a vector counting the number of objects identified per image. This representation was called “counting representation”. It has the advantage of being generally less sparse than the “one-hot representation”. Despite of that, it still has many zeros, as not every image has every class.

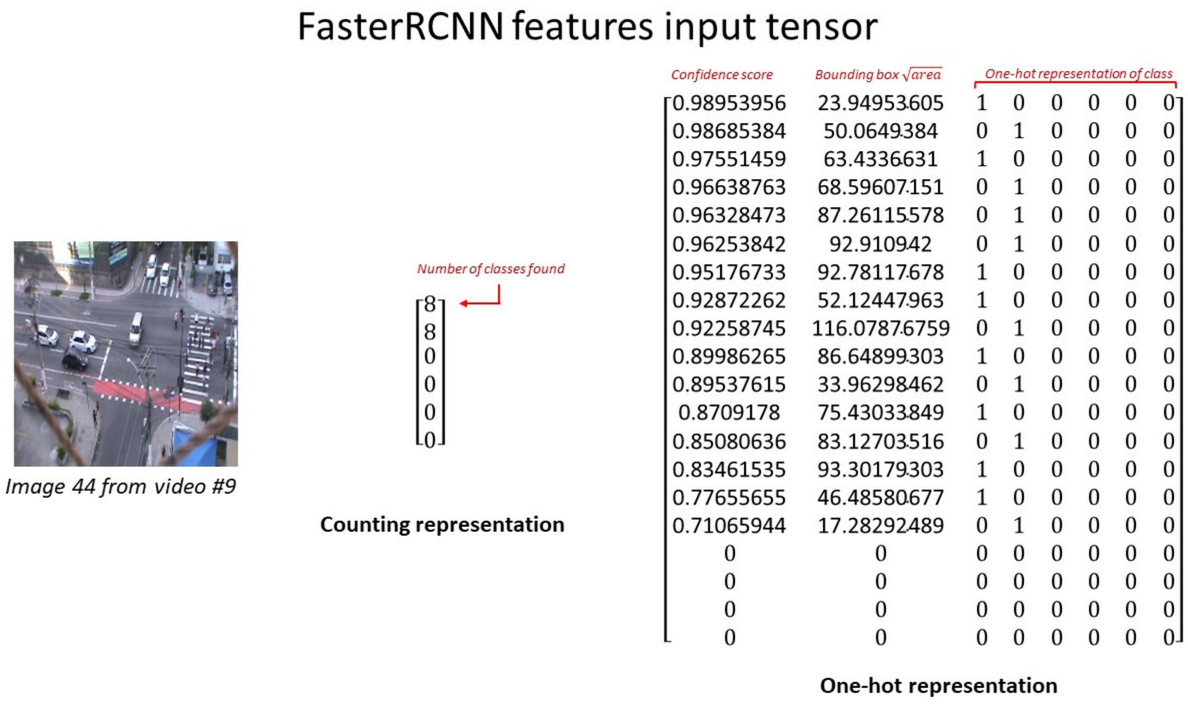


Figure 3.8: Faster R-CNN features tensor.

3.3 Model Evaluation

Data evaluation was briefly mentioned in Section 2.3.2. Two figures of merit were used in order to classify the models: the mean correlation between folds and the mean MSE between folds. Both these figures are calculated between the predicted models output and its correspondent datasets target output. The lower the MSE and the higher the correlation, the better is the model.

Using the mean MSE and correlation between folds allows to mitigate particularities on each dataset, having a better understanding of the architectures performance on general data.

Chapter 4

Results

In this chapter, results from all our training sessions are reported. Training sessions were divided into three sections:

- Hyperparameter Choise

  A variety of models are first trained over one unique fold. In this training session, the aim is to reduce the number of hyperparameters to be trained over multiple folds, selecting only the most relevant hyperparameters variation. This allows the training of a reduced number of variation over multiple folds.

- Cross-fold Training

  After selecting which hyperparameters are the most relevant in the analysis, an experiment of training all remaining network variations over the ten proposed folds was conducted. With this, the mean behaviour of each architecture over the proposed problem is anylised.

- Usage of Regularization

  Finally, the use of two different regularization methods, over the best temporal and non-temporal architecture, over all folds and 300 epochs per model was tested.

Results for each training session are reported using always its best epoch (epoch with the minimum MSE) on validation data. Training data results are omitted. In all reported results, one frame equals a time step of approximately 1/3 of a second, unless stated otherwise.

4.1 Hyperparameter Choice

As stated in Subsection 3.2.2, there is a great number of hyperparameters to be considered for network training. Running all possible networks proves to be unfeasible. Instead, a smaller training over one fold was conducted in order to determine what are the most relevant hyperparameters for LSTM and non-LSTM architectures.

In this smaller test, hyperparameters for both LSTM and non-LSTM networks are:

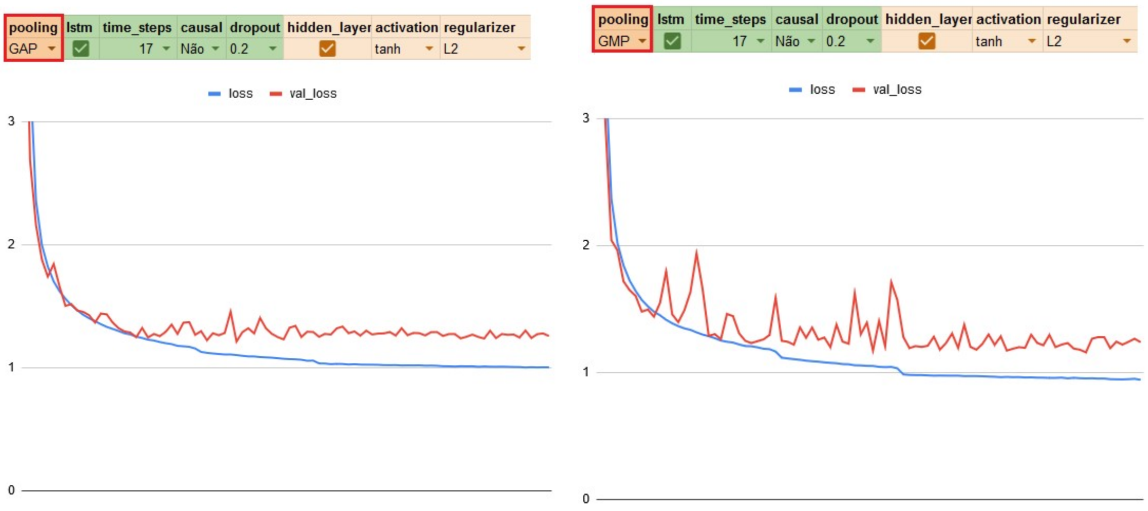
- Extractor: VGG16.
- Used pooling layer: GMP or GAP layer.
- LSTM layer: number of time steps on LSTM (9 or 17), usage of causal or non-causal dataset on LSTM, usage of dropout on LSTM (0%, 20% or 50%).
- Usage of a hidden FC layer on model: activation function of hidden FC layer (relu or tanh) and usage of L2 regularization on hidden FC layer (L2 or none).

4.1.1 Reported Results for First Training

Results from this first test did not show significant improvements on using GMP over the GAP pooling layer. GAP also shows to have a better training curve of loss over epochs, as demonstrated on Figure 4.1.

This may be explained by seeing the global average pooling as a structural regularizer [23].

Not using the LSTM layer in this first test yielded noisier results on time prediction for every LSTM counterpart, as seen in Figure 4.2. Despite of that, training loss for both LSTM and non-LSTM models were close.



pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GAP	☑	17	Não	0.2	☑	tanh	L2

pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GMP	☑	17	Não	0.2	☑	tanh	L2

(a) GAP training loss.

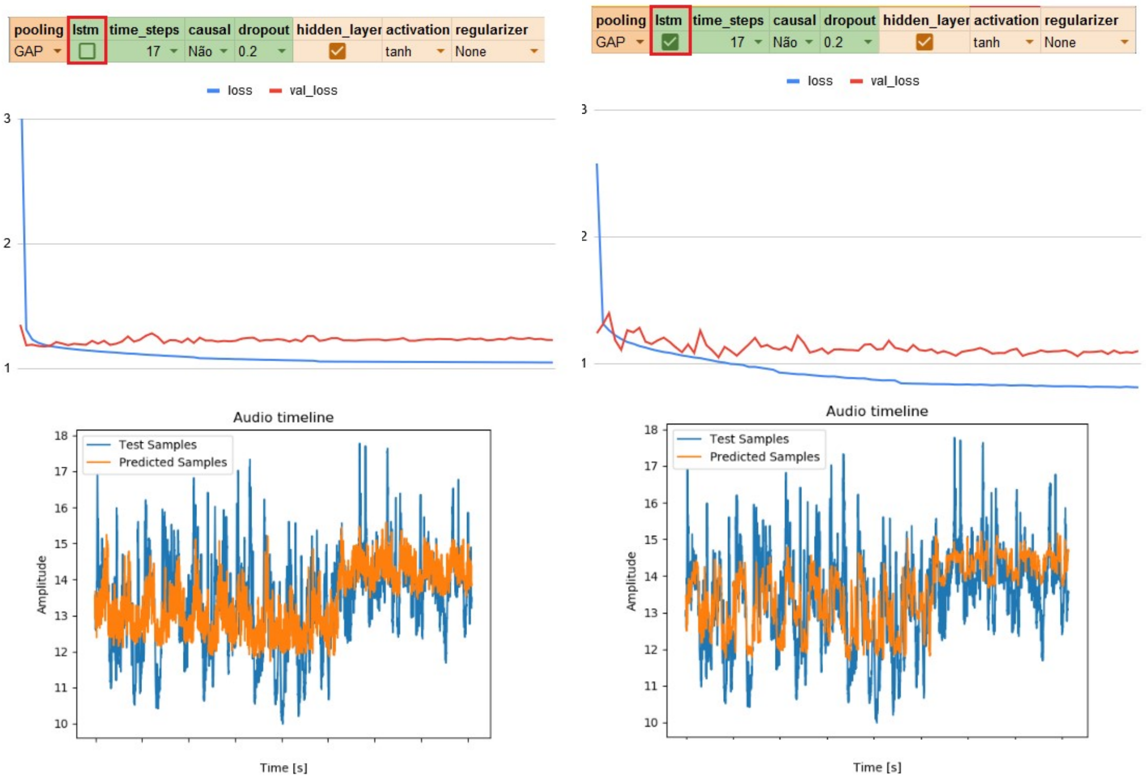
(b) GMP training loss.

Figure 4.1: Training loss over 90 epochs. Loss plot over epochs present larger spikes when using GMP: (a) GAP training loss; (b) GMP training loss.

Usage of 17 frames (5,66 seconds of video) usually yields better minimization results when compared with using 9 frames (3 seconds of video). The usage of a non-causal target prediction on LSTM is also consistently better than using a causal model. Usage of dropout on the LSTM layer proves to be essential. Not using dropout consistently overfitted the model. There is no significant difference between using 20% or 50% dropout.

Changing the activation function of the hidden FC layer between relu and the hyperbolic tangent does not show any significant differences. The usage of L2 regularization on the FC layer shows to slow down the training process. This produced a slower training curve on all network configurations. Nevertheless, best epoch on models without FC regularization yields satisfactory results despite the overfitting on latter epochs, as depicted in Figure 4.3. Regularization was key on not overfitting models without LSTM.

After these tests, the usage of the global average pooling layer was chosen over the global max pooling for its consistent results and structure regularization; on expanding the tests from 17 to 32 frames, using 9 and 32 frames input; opting for a non-causal dataset with 20% dropout on LSTM; and on using the hyperbolic tangent activation with no regularization in the FC layer. The non-usage of regularization in the hidden FC layer was opted, as it would extend considerably the necessary



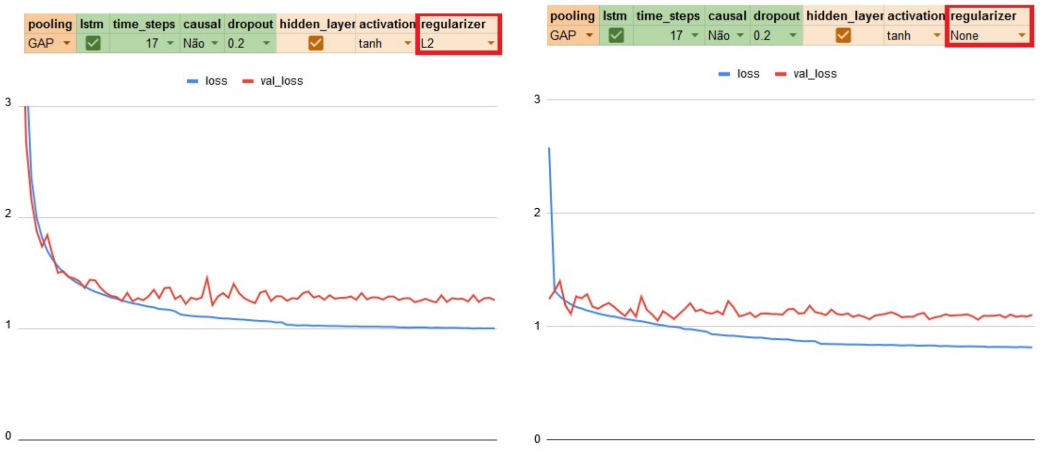
pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GAP	☐	17	Não	0.2	☑	tanh	None

pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GAP	☑	17	Não	0.2	☑	tanh	None

(a) LSTM model.

(b) non-LSTM model.

Figure 4.2: Difference between using LSTM and not using LSTM layer: (a) LSTM models training loss and time prediction; (b) non-LSTM models training loss and time prediction.



pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GAP	☑	17	Não	0.2	☑	tanh	L2

pooling	lstm	time_steps	causal	dropout	hidden_layer	activation	regularizer
GAP	☑	17	Não	0.2	☑	tanh	None

(a) LSTM with L2 regularization.

(b) LSTM with no regularization.

Figure 4.3: Difference between using L2 regularizer and no regularizer: (a) LSTM with L2 regularization; (b) LSTM with no regularization.

training time for all networks. Instead, the usage of regularization was tested after, on the best LSTM and non-LSTM models from the cross-fold training session only.

4.2 Cross-fold Training

In this training session, multiple network variations were tested over ten folds in order to measure the network mean behaviour over folds. Tested hyperparameters were:

- Convolutional network: VGG16, ResNet50 and InceptionV3.
- LSTM layer: 9 and 32 frames input.
- Usage of Hidden FC layer on model.
- Usage of an auxiliary input on model.

Detailed list of all trained models is available on Appendix B.

4.2.1 Results for non-LSTM Models

Within the non-LSTM networks, the best individual result for both best and last epoch MSE was archived by network 13 (ResNet50 with hidden-layer) on fold 9, as seen on Table 4.1.

Table 4.1: 10 best individual results for non-LSTM networks.

Fold	Model	Best epoch MSE	Last epoch MSE
9	13	1.13	1.32
9	12	1.22	1.46
9	16	1.22	1.33
9	14	1.27	1.98
4	12	1.28	1.52
8	13	1.32	1.74
6	12	1.34	1.86
4	15	1.34	1.34
4	16	1.37	1.39
4	13	1.38	2.73

However, cross-fold validation is necessary when analysing a model. Best mean MSE for all folds was archived by model 12 (VGG16 with hidden-layer), as seen on Table 4.2.

Table 4.2: non-LSTM mean MSE, ordered by best epoch MSE.

Model	Mean Best Epoch MSE	Mean Last Epoch MSE
12	1.46	2.10
15	1.56	1.60
13	1.58	2.26
16	1.68	1.87
14	2.06	2.76
17	2.46	2.76

It is interesting to notice that, by ordering these results by last epoch MSE, models without a hidden layer (15, 16, 17) surpass its equivalent with a hidden-layer (12, 13, 14). This result suggests that the usage of a hidden-layer in non-LSTM networks can produce better results, however making the network more prone to overfitting, as shown in Figure 4.4. Some time predictions can also be seen in Figure 4.5.

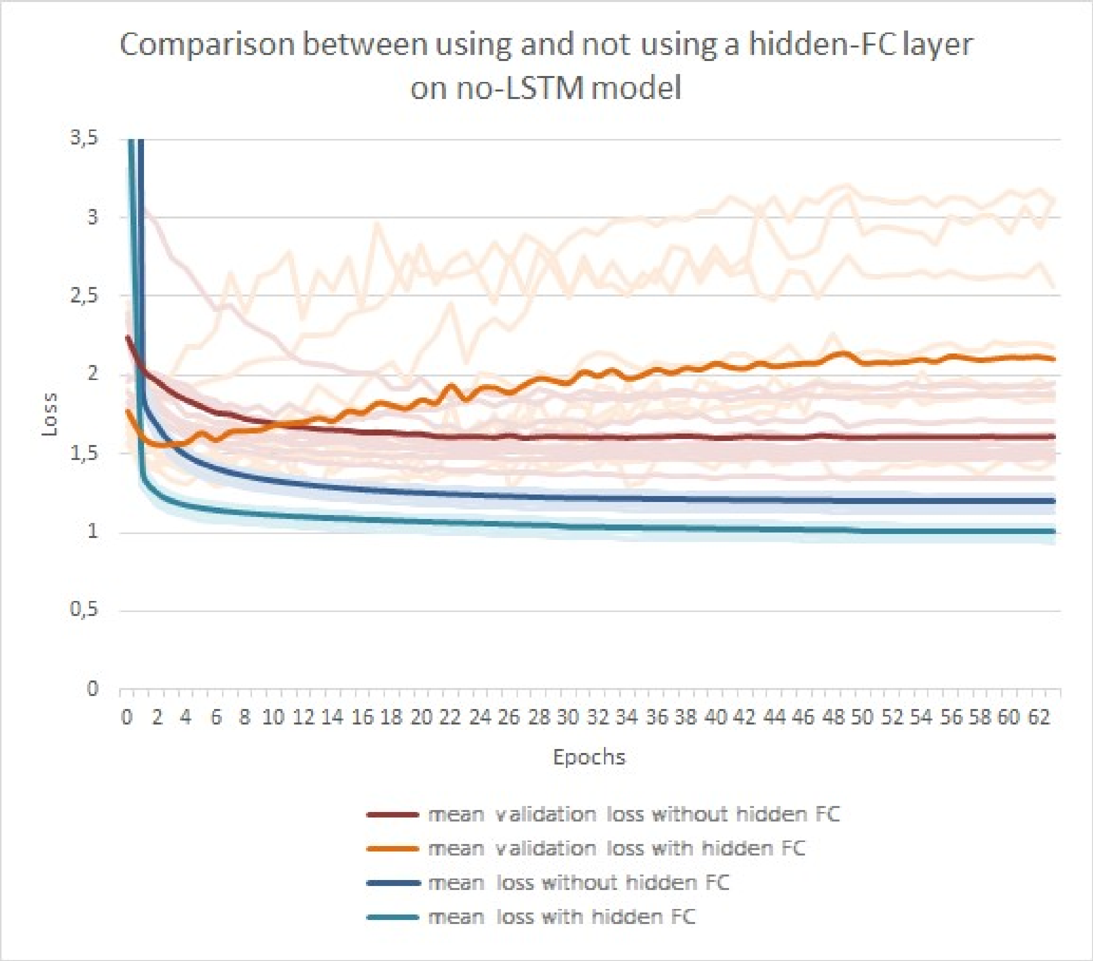
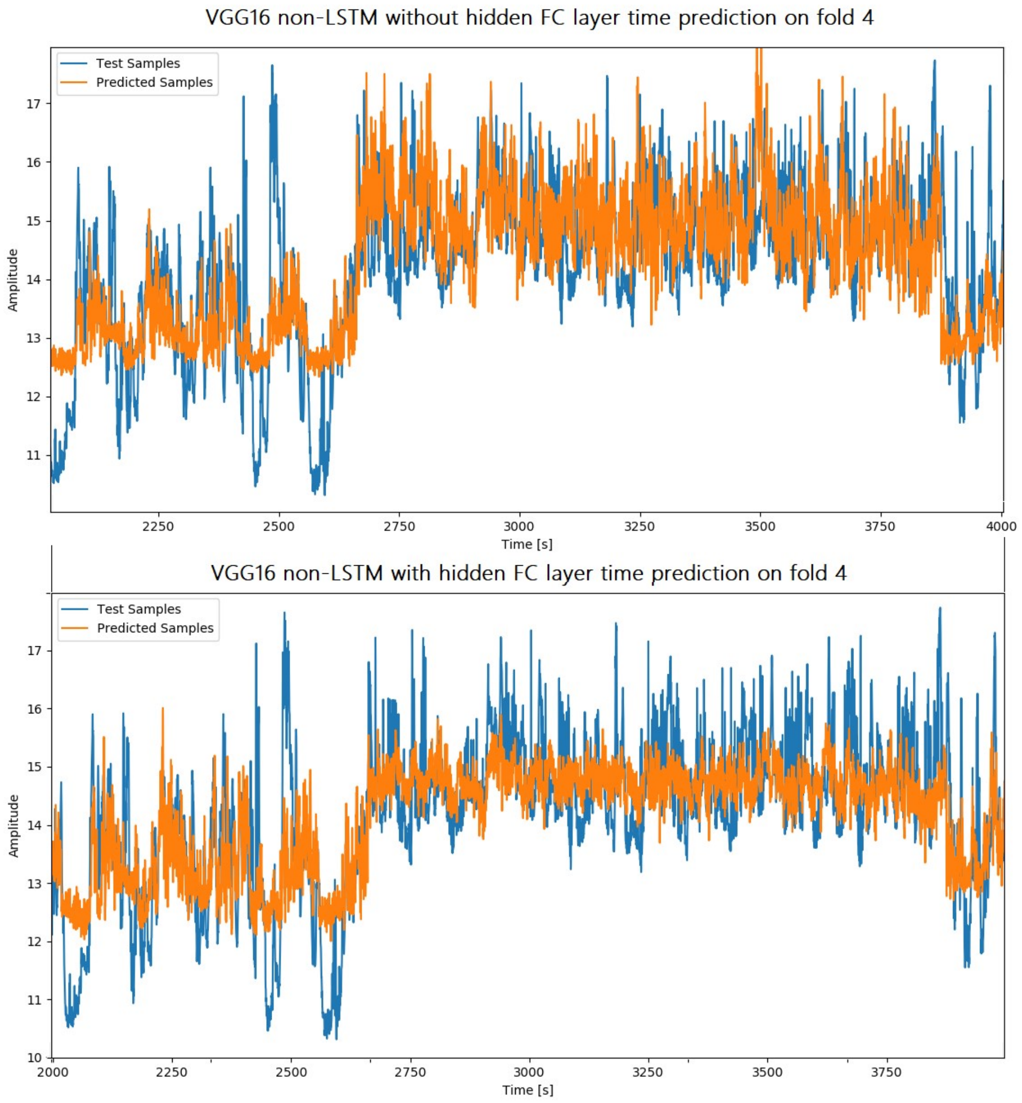


Figure 4.4: Comparison between using and not using a hidden FC layer on non-LSTM models (VGG16). Clear lines are the training curves for individual folds. It is noticeable that the hidden-layer makes the model learn faster and better at the initial epochs, but it easily over-fits on the following epochs.

It is also noticeable that the VGG16 obtains the best results, followed by the ResNet50 and the InceptionV3. These results match with what was reported in previous research [9].



VGG16 non-LSTM without hidden FC layer time prediction on fold 4

VGG16 non-LSTM with hidden FC layer time prediction on fold 4

Figure 4.5: VGG16 non-LSTM time predictions with and without the hidden FC layer. Results from the last-epoch of training on fold 4.

When comparing the three used convolutional networks structures, the VGG16 has the least tendency to overfit the models hidden layer, and generally yields better results. The InceptionV3 network is consistently the worse between the three networks. This behaviour is shown in Figure 4.6.

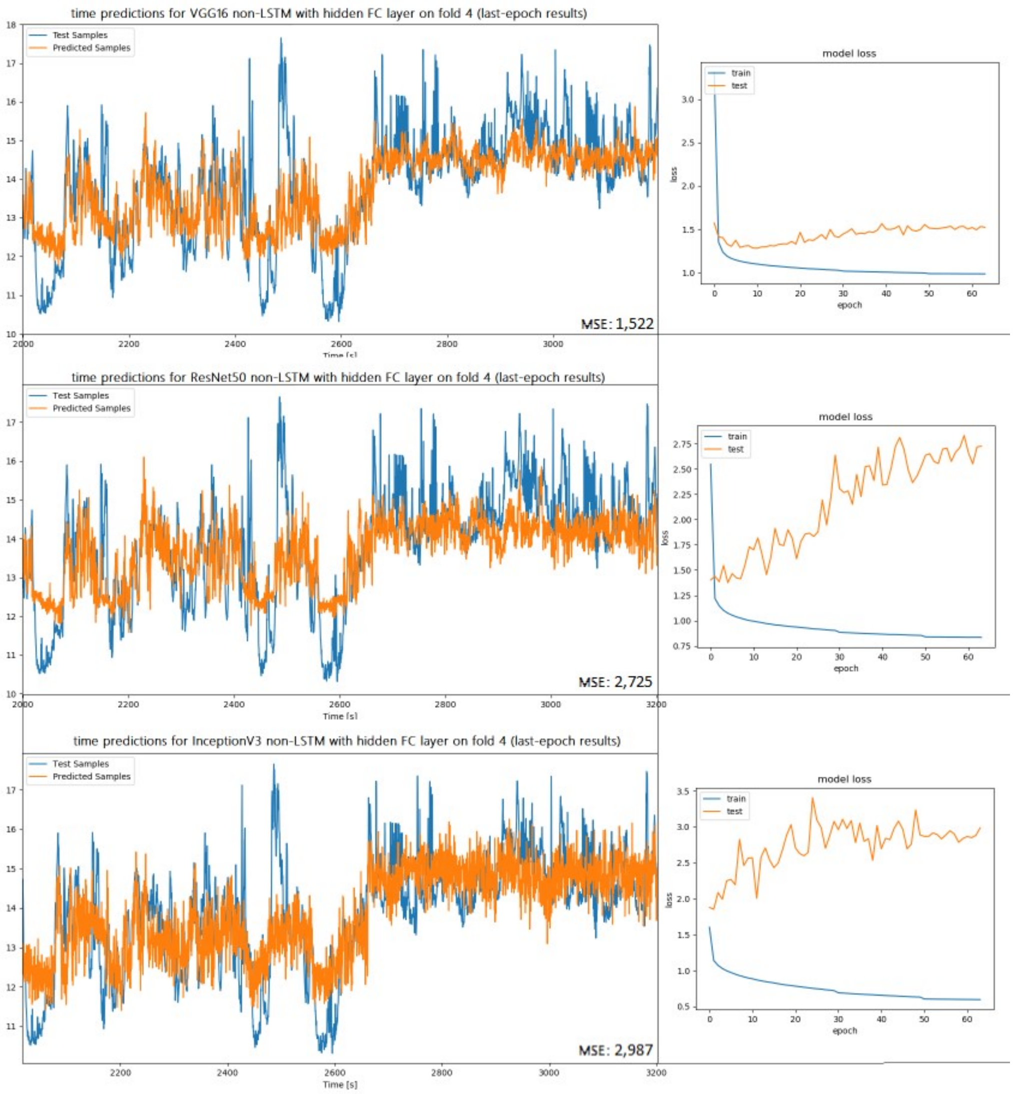


Figure 4.6: Comparison between VGG16, ResNet50 and InceptionV3 based non-LSTM models with hidden FC layer and last-epoch time prediction on fold 4. We can see the effects of overfitting the hidden-layer has in these networks.

The models performance were also analysed using the Pearson correlation, as shown in Table 4.3.

Table 4.3: Mean correlation for non-LSTM models.

Model	Mean Best Epoch Correlation	Mean Last Epoch Correlation
12	59.73%	49.87%
15	57.62%	57.83%
13	57.29%	47.53%
16	55.15%	51.78%
14	41.83%	34.59%
17	32.84%	31.17%

These numbers match with the MSE results. The model with the best correlation coefficient for best epoch is the VGG16 with hidden-layer, followed next by VGG16 without hidden-layer.

In all tests, the ResNet50 and InceptionV3 networks were respectively in second and third places.

4.2.2 Influence of auxiliary input on non-LSTM models

The usage of an auxiliary input on non-LSTM models are reported. Slightly better results for best epoch mean MSE were achieved for validation data on all models with an auxiliary input, as shown in Table 4.4.

The best last-epoch MSE still goes to model 15 (VGG16 without hidden-layer and no auxiliary input). This once more shows that the models hidden FC layers on both the auxiliary and main input sides have a strong tendency to overfit during training sessions, and might need some stronger regularization methods.

In regards to the representation techniques used, the “counting representation” achieved consistently better results in all networks compared to the “one-hot representation”. This suggests that although the “one-hot representation” has more data, its larger sparsity and structure complexity harms the training process. It also implies that the number and class of vehicles may be the most important information to be processed by the network.

Table 4.4: Mean MSE for all non-LSTM models, including models with an auxiliary input.

model	mean best epoch MSE	mean last epoch MSE
48	1.44	2.24
51	1.45	2.19
12	1.46	2.10
15	1.56	1.60
49	1.56	2.45
52	1.57	2.35
13	1.58	2.26
16	1.68	1.87
50	1.90	3.34
53	1.96	3.12

4.2.3 Results for LSTM models

Results for LSTM models are reported. Almost all models that used an LSTM network and converged proved to have an smaller MSE and better correlation than its non-LSTM counterpart. Best individual results for best epoch MSE on train set goes to models 7 (VGG16 with 32 frames input and no hidden FC layer) on fold 9. The 10 best individual results are seen in Table 4.5.

However cross-fold validation is necessary to analyse the best model on all folds. Model with best mean performance is shown in Table 4.6

It can be seen that model 1 (VGG16 with 32 frames LSTM input and hidden FC layer) is the best model, followed by model 7 (VGG16 with 32 frames LSTM input and no hidden FC layer). Those are followed by models 6 and 0, both of then VGG16 with 9 frames LSTM input.

From this result, three aspects can be addressed.

- The VGG16 once more shows to be the best convolutional architecture for this application.
- Longer LSTM sequences shows to yield better prediction results.

Table 4.5: Best individual models for LSTM networks without an auxiliary input.

Fold	Model	Best Epoch MSE	Last Epoch MSE
9	7	0.929	0.385
6	1	0.934	0.415
7	1	0.961	0.376
9	1	0.970	0.440
6	7	1.011	0.486
9	9	1.018	0.468
9	3	1.045	0.486
9	6	1.047	0.474
9	2	1.056	0.498
9	8	1.057	0.498

Table 4.6: LSTM models with best mean MSE, without an auxiliary input.

Model	Mean Best Epoch MSE	Mean Last Epoch MSE
1	1.15	1.27
7	1.21	1.34
6	1.40	1.99
0	1.42	1.93
3	1.51	1.83
18	1.52	2.30
25	1.54	1.90
9	1.55	1.86
24	1.56	1.99
26	1.61	2.13

- The regularization technique used in the LSTM (20% dropout) shows to also influence the following hidden FC layer. Tested LSTM models did not suffer with overfitting as much as non-LSTM based models.

Some time prediction samples for these networks can be seen in Figure 4.7.

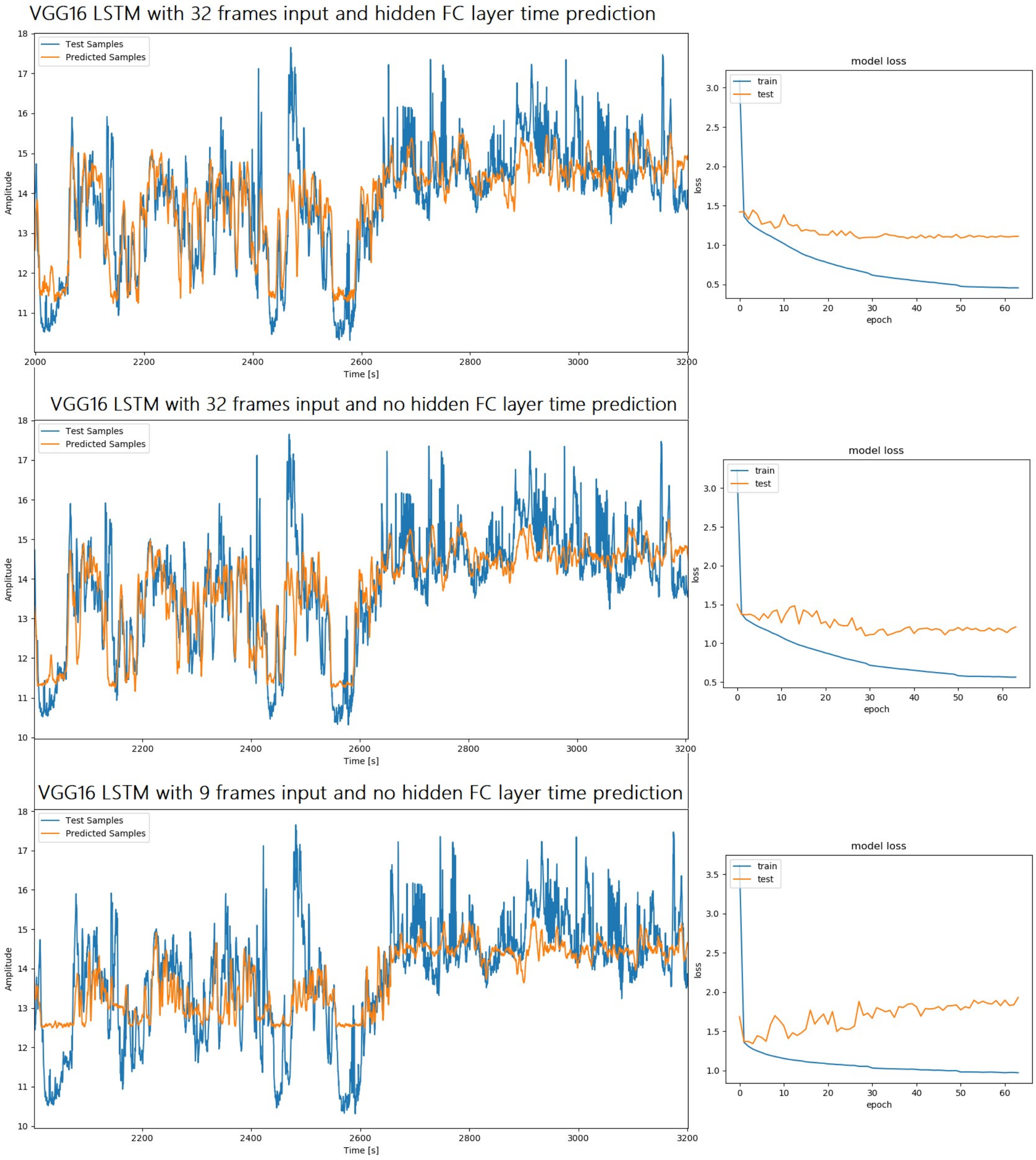


Figure 4.7: Best individual models for LSTM networks without an auxiliary input. Predictions over fold 4, best epoch.

It is noticeable that the LSTM models provides a less noisy signal compared to its non-LSTM counterpart. This is discussed in details in Section 4.4.

Models were also analysed in terms of its mean correlation. 10 models with best correlation are shown in Table 4.7.

Table 4.7: LSTM models ordered by mean best epoch correlation.

Model	Mean Best Epoch Correlation	Mean Last Epoch Correlation
1	70.53%	68.22%
7	69.45%	67.64%
6	62.24%	54.02%
9	61.41%	56.75%
3	61.04%	57.52%
18	59.61%	46.50%
25	58.48%	54.63%
2	58.24%	48.31%
8	58.15%	51.71%
24	58.01%	51.23%

These results agree with those of the MSE. Models 1 and 7 (VGG16 32 frames input LSTM) reach above 69% mean correlation levels.

Models that used an LSTM at output (models 18 to 29), tend to easily diverge during training. An example of a time model that did not converge is shown in Figure 4.8. Models that used a stateful LSTM network implementation (models 30 to 41) also did not converge.

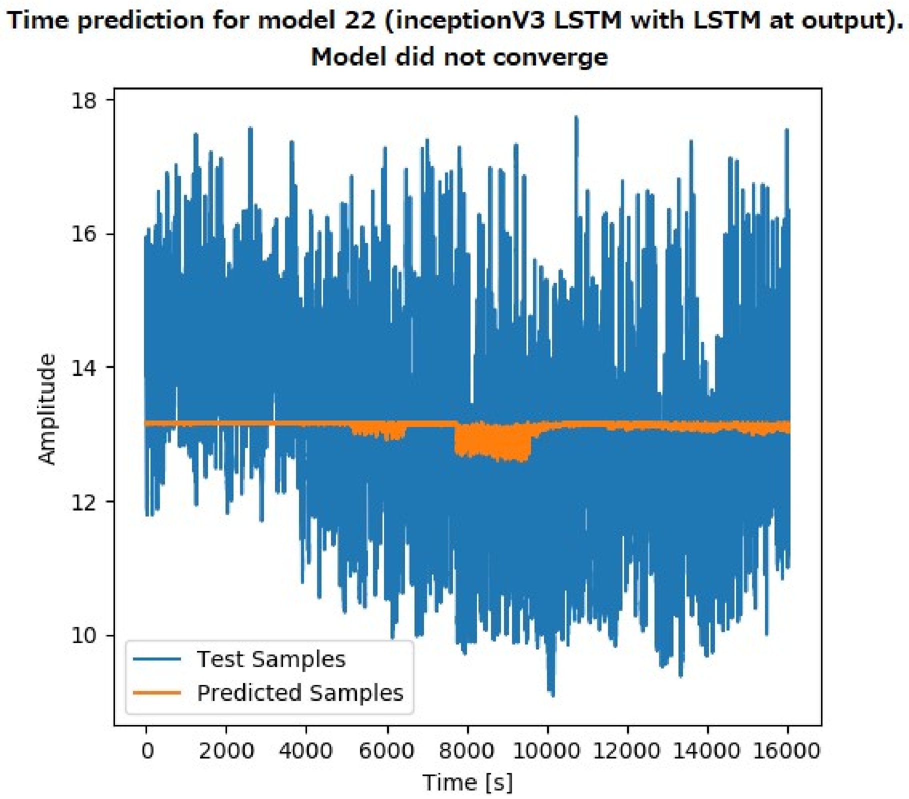


Figure 4.8: Time prediction for model 22 on fold 0. We can see that this model did not converge.

4.2.4 Influence of auxiliary input on LSTM models

Considering now LSTM models with an auxiliary features input, the exact same pattern is seen as in Section 4.2.2. VGG16 based LSTM models that used an auxiliary features input obtained slightly better results both in terms of mean MSE, depicted in Table 4.8, and mean correlation, shown in Table 4.9, with an above 70% correlation.

Table 4.8: LSTM models ordered by best epoch mean MSE.

Model	Mean Best Epoch MSE	Mean Last Epoch MSE
42	1.141	1.262
45	1.153	1.250
1	1.155	1.266
7	1.214	1.338
6	1.399	1.987
46	1.410	1.678
0	1.416	1.932
43	1.427	1.814
48	1.437	2.244
51	1.451	2.190

Table 4.9: LSTM models ordered by best epoch mean correlation.

Model	Mean Best Epoch correlation	Mean Last Epoch correlation
42	70.99%	68.52%
45	70.76%	68.83%
1	70.53%	68.22%
7	69.45%	67.64%
46	63.49%	59.53%
43	62.57%	57.90%
6	62.24%	54.02%
48	61.85%	44.82%
9	61.41%	56.75%
51	61.12%	45.28%

In terms of features representation strategies, once more the “counting representation” yields better results than the “one-hot representation”, asserting what was discussed in Section 4.2.2.

After showing the positive impacts of an auxiliary input in the model, the experiments using classification networks were halted, and understood as future works.

4.3 Regularization on Temporal and Non-Temporal Models

As a last experiment, the effects of two different regularization methods on the hidden FC layer of the best LSTM and non-LSTM models were tested, without exploring the effects of using an auxiliary faster R-CNN input. In this experiment, models 12 and 1 (VGG16, hidden FC layer, respectively with and without LSTM layer) were trained over 300 epochs and ten folds, using the following regularization methods:

- 20% *Dropout*.
- L2 regularizer with Keras (refer to Page 27) default parameters.

Results are shown in Figures 4.9, 4.10, 4.11 and 4.12. In all figures: blue lines represents the loss figure over epochs on the training dataset; orange lines represents the loss figure over epochs on the test dataset; and clear lines represent the individual figures for each individual fold, while the darker lines represent the mean figures between folds.

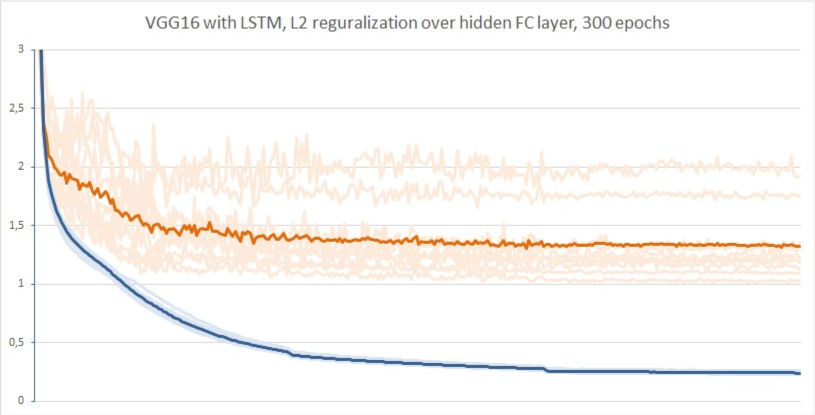


Figure 4.9: Model 60: Best LSTM model with L2 regularization over 300 epochs.

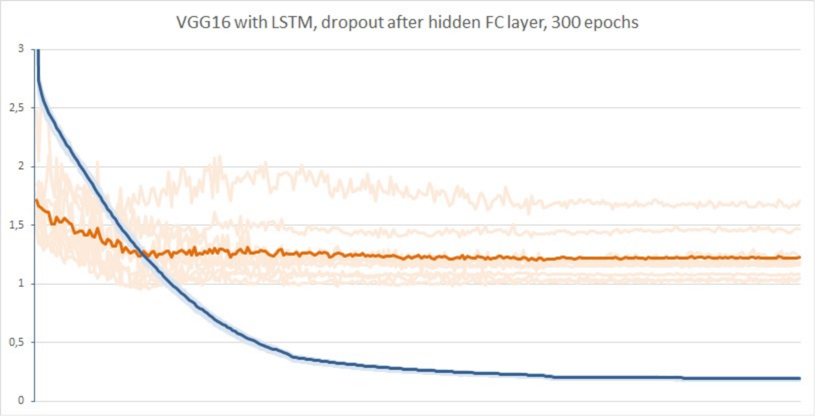


Figure 4.10: Model 61: Best LSTM model with dropout regularization over 300 epochs.

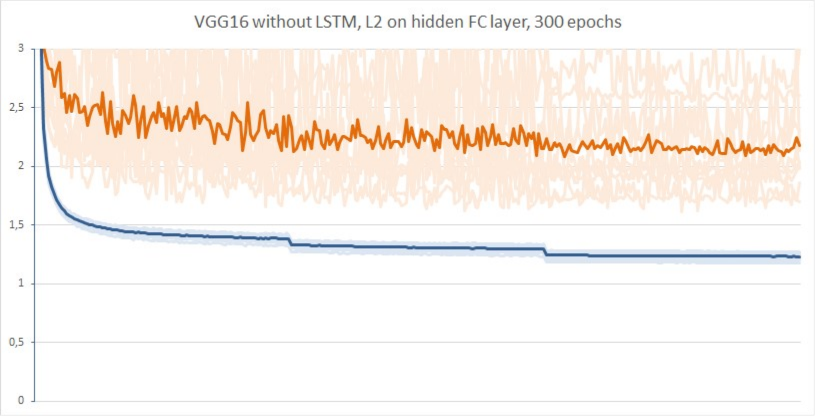


Figure 4.11: Model 62: Best non-LSTM model with L2 regularization over 300 epochs.

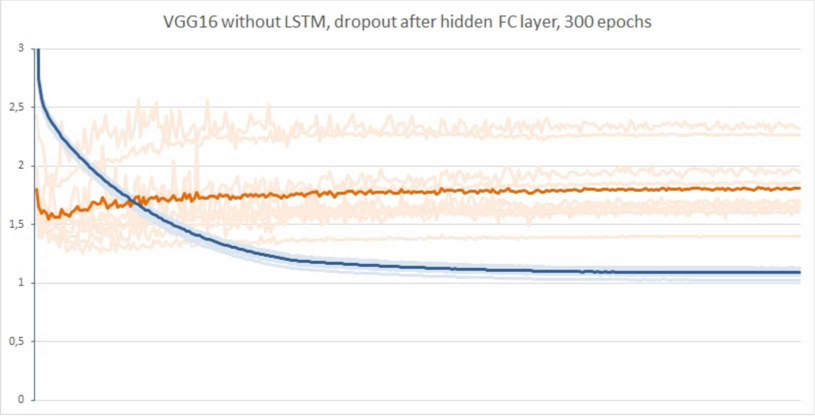


Figure 4.12: Model 63: Best non-LSTM model with dropout regularization over 300 epochs.

Tables 4.10 and 4.11 shows that, in this experiment, the dropout after the hidden FC layer is the best regularization method for both LSTM and non-LSTM architectures.

Table 4.10: Best non-LSTM and LSTM models MSE, validation data.

Model	Mean Best Epoch MSE	Mean Last Epoch MSE
LSTM, L2	1.19	1.29
LSTM, Dropout	1.14	1.22
non-LSTM, L2	1.74	2.01
non-LSTM, Dropout	1.46	1.80

Table 4.11: Best non-LSTM and LSTM models correlation, validation data.

Model	Mean Best Epoch Correlation	Mean Last Epoch Correlation
LSTM, L2	69.22%	67.04%
LSTM, Dropout	71.34%	68.76%
non-LSTM, L2	53.33%	52.06%
non-LSTM, Dropout	60.19%	54.16%

4.4 Ablation study: LSTM vs non-LSTM models

In this section, the usage of LSTM layers in the model is compared with not using LSTM layers.

4.4.1 Model and training complexity

LSTM models requires more resources from hardware in order to be trained. On Keras, input tensor for the LSTM layer is of shape "batch size×time steps×features". Tensor shape for non-LSTM models is simply "batch size×features", as show in Figure 4.13. This means that the amount of memory necessary for non-LSTM models is multiplied by the additional time steps dimension on LSTM models (If using a sliding window for frame input).

The additional layer also multiplies the amount of trainable parameters. This makes training process slower for LSTM models. In this work, LSTM models took from 60 to 150 seconds per epoch to train upon, while non-LSTM models took approximately 6 to 15 seconds. In practice, LSTM models showed to require 8 to 10 times more time to finish training.

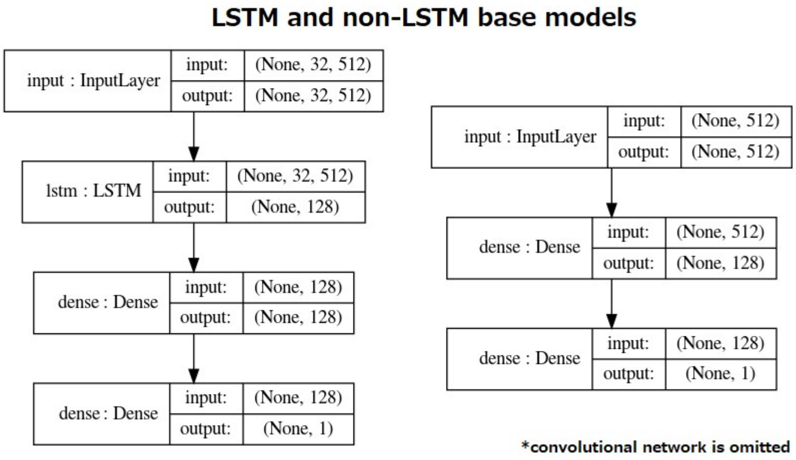


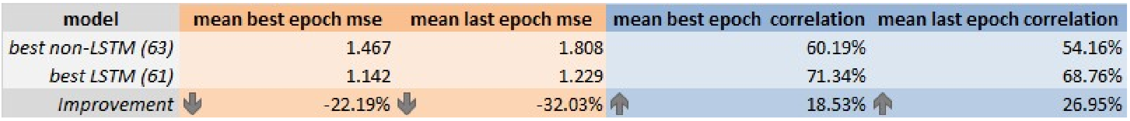
Figure 4.13: LSTM and non-LSTM base models structure (VGG16 extractor).

4.4.2 MSE and Correlation differences

As previously shown, LSTM models shows a significant improvement in terms of MSE and correlation figures.

A direct comparison of the best non-LSTM model and LSTM models (without auxiliary input, dropout regularization on the hidden FC layer) is showed on Table 4.12.

Table 4.12: Comparison of best non-LSTM and LSTM, without auxiliary input models.

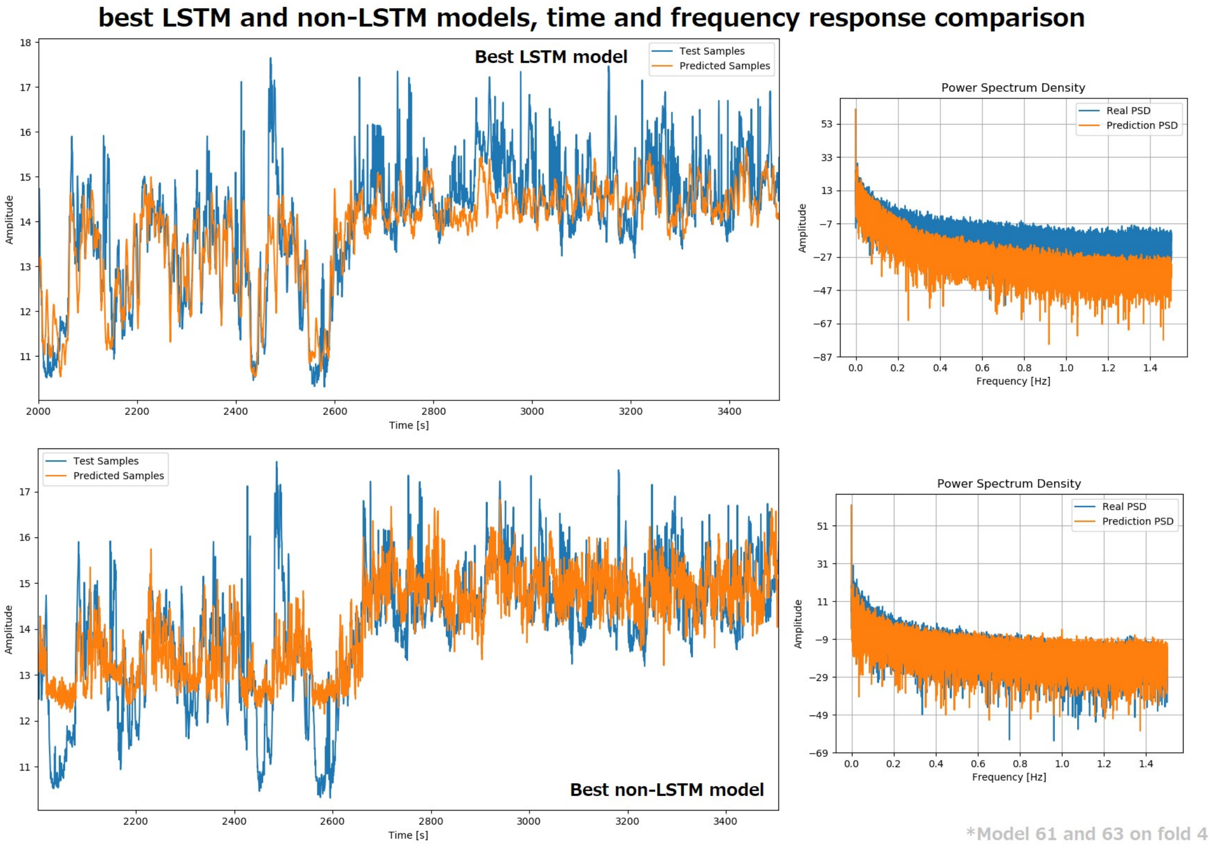


model	mean best epoch mse	mean last epoch mse	mean best epoch correlation	mean last epoch correlation
best non-LSTM (63)	1.467	1.808	60.19%	54.16%
best LSTM (61)	1.142	1.229	71.34%	68.76%
Improvement	⬇ -22.19%	⬇ -32.03%	⬆ 18.53%	⬆ 26.95%

It can be seen that the LSTM models have a correlation 18.5% higher compared to non-LSTM models (best epoch). Minimized MSE is also 22.2% smaller on LSTM models (best epoch). These results objectively show how the LSTM layer improves the prediction results. LSTM models also show a smaller tendency to overfit, even after applying the dropout layer on the model.

4.4.3 Time and frequency response

Time prediction of non-LSTM models showed to be noisier than its LSTM counterpart. This can be seen on its output time series. When calculating the power spectrum density (PSD) for both LSTM and non-LSTM models, we notice that the PSD for the non-LSTM model follow the PSD of the original signal, while the LSTM filters high frequencies. We can conclude that the LSTM layer behaves as a low-pass filter, as depicted in Figure 4.14.



*Model 61 and 63 on fold 4

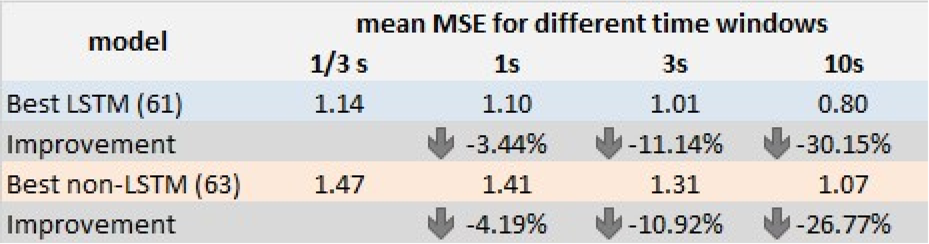
Figure 4.14: Comparison of best non-LSTM and LSTM, without auxiliary input models. Image show the time response and power spectrum density of the signals.

4.5 Window Length Analysis

In this last section, the mean behaviour of the best temporal and non-temporal architectures here proposed are analysed over different time windows. All results until now presented correspond to instantaneous time window of 1 frame (1/3 seconds).

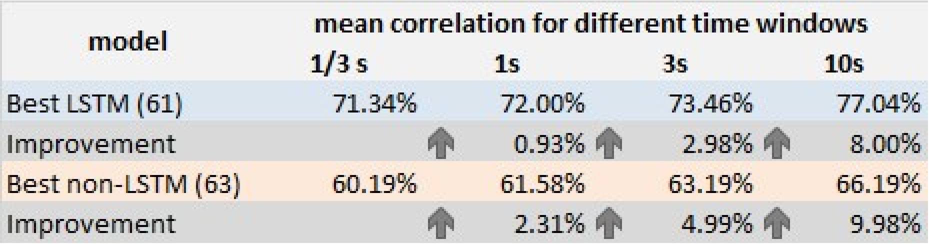
In order to estimate how these networks perform over different time windows, a moving average filter (see Section 2.3.2) was used to average both the target and predicted sound pressures, and recalculate the MSE and correlation figures. The results are reported using windows of 3, 9 and 30 frames, corresponding to time windows of 1, 3 and 10 seconds of video. Some examples of this filtering can be seen in Figure 4.15. The improvement measured from such filtering in terms of MSE and correlation figures are shown respectively in Tables 4.13 and 4.14.

Table 4.13: MSE behaviour over different time windows.



model	mean MSE for different time windows			
	1/3 s	1s	3s	10s
Best LSTM (61)	1.14	1.10	1.01	0.80
Improvement		⬇ -3.44%	⬇ -11.14%	⬇ -30.15%
Best non-LSTM (63)	1.47	1.41	1.31	1.07
Improvement		⬇ -4.19%	⬇ -10.92%	⬇ -26.77%

Table 4.14: Correlation behaviour over different time windows.



model	mean correlation for different time windows			
	1/3 s	1s	3s	10s
Best LSTM (61)	71.34%	72.00%	73.46%	77.04%
Improvement		⬆ 0.93%	⬆ 2.98%	⬆ 8.00%
Best non-LSTM (63)	60.19%	61.58%	63.19%	66.19%
Improvement		⬆ 2.31%	⬆ 4.99%	⬆ 9.98%

It is noticeable that by averaging the sound pressure over longer periods of time, a better correlation and lower MSE figures can be obtained, which is expected. This is a useful statement when instantaneous sound pressure is not needed. With this approach, the network is not retrained using the filtered dataset (only the dataset output is filtered).

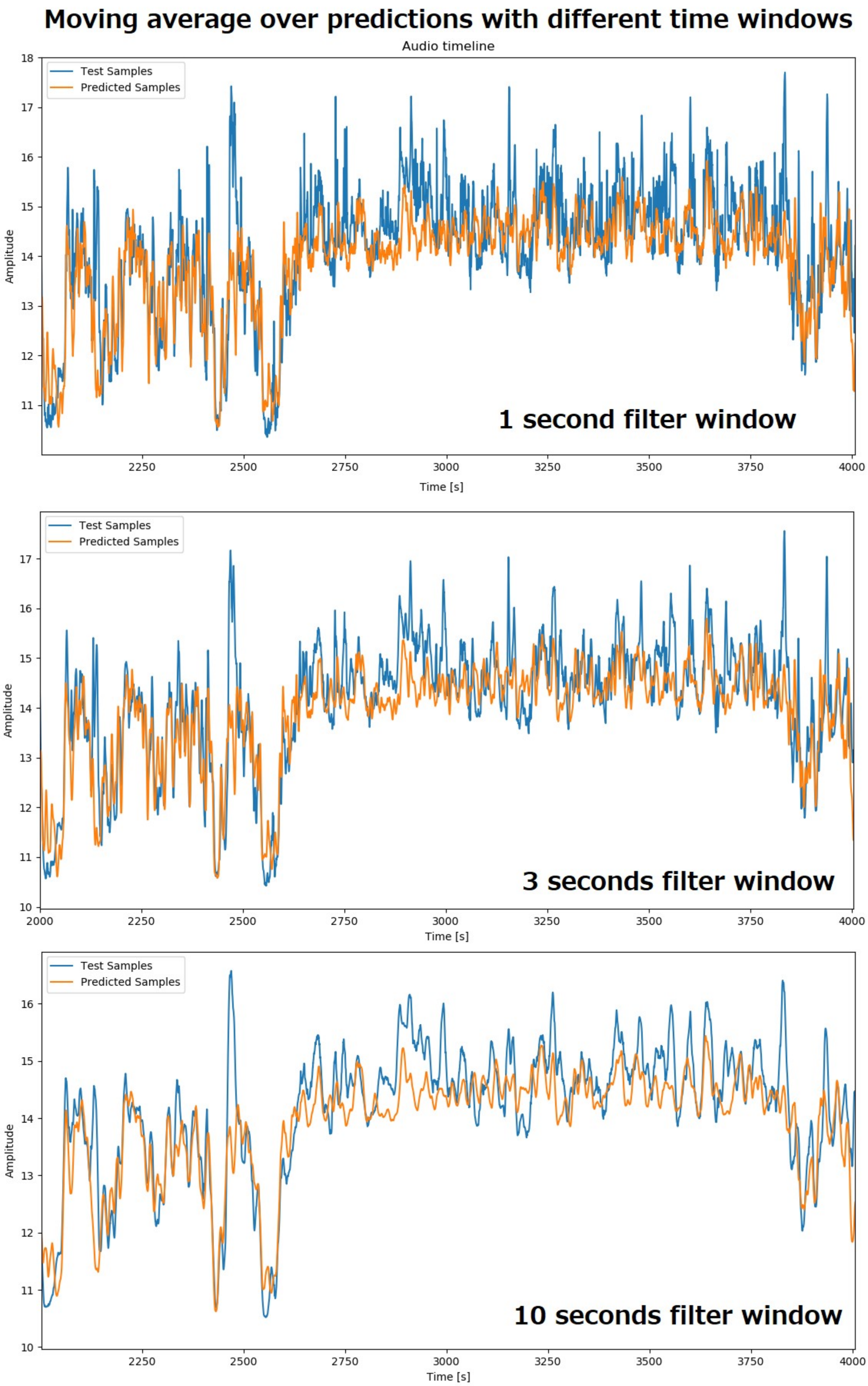


Figure 4.15: Predictions behaviour over different time windows for best LSTM model (model 61). Time predictions on fold 4.

Chapter 5

Conclusion

In this research, temporal models consistently outperform non-temporal models. It was shown how the usage of longer sequences with a non-causal dataset loading is preferable. The best results for single-input models was achieved by model 61, detailed in Figure 5.1. The usage of auxiliary inputs were explored, using features extracted with a Faster R-CNN network in two different tensor representations. It was shown how dual input models can slightly improve predictions. It was reported how the tested models tend to overfit, and two different methods of regularization were tested, opting for the dropout method.

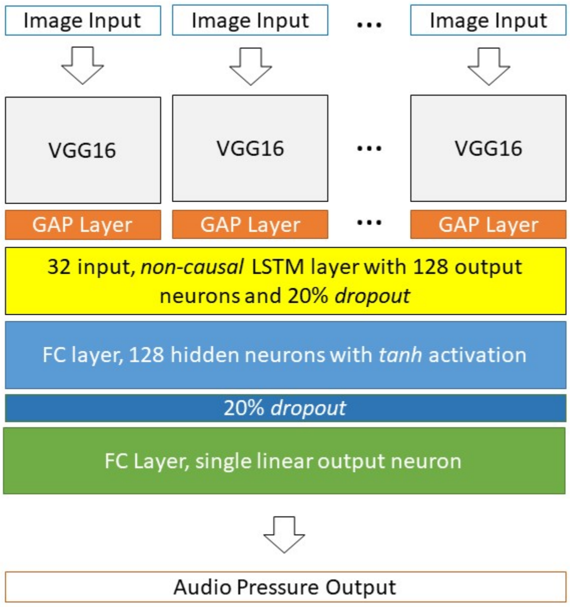


Image Input
Image Input
...
Image Input
VGG16
VGG16
...
VGG16
GAP Layer
GAP Layer
...
GAP Layer
32 input, *non-causal* LSTM layer with 128 output neurons and 20% *dropout*
FC layer, 128 hidden neurons with *tanh* activation
20% *dropout*
FC Layer, single linear output neuron
Audio Pressure Output

Figure 5.1: Detailed view of the best single-input model.

5.1 Future Works

During this research, a number of possible future works were identified. A summarizing of the most promising ideas are presented as follows.

- The creation of a new dataset, containing images from multiple angles and places. Using images from real CCTV cameras is preferable. Also, recording of audio closer to the framed scene is interesting. This will allow the model to be tested closer to a real scenario. Closer inspecting how time of the day interfere in the prediction results (Are predictions at night easier and better than at day?).

- Closer exploration of the convolutional layer. This research was made using the simplest way of integrating temporal features in a video analysis. There are however, more sophisticated convolutional models, such as 3D convolutional blocks [21, 24] and Convolutional LSTMs [25]. This will require the training of the convolutional network, and may improve results even further.

- The usage of two input models showed promising results, and require a closer study. The integration of classification networks such as the Faster R-CNN or the YoloV4 into the analysis might be the best way of improving results.

- Development of a visualization tool integrating video cameras, sound maps and traffic reports is interesting, as it puts this and related research closer to a practical implementation of given system for smart cities.

Bibliography

[1] GOINES, L., HAGLER, L., "Noise pollution: A modern plague", 2007.

[2] MA, J., LI, C., KWAN, M. P., *et al.*, "A multilevel analysis of perceived noise pollution, geographic contexts and mental health in Beijing", *International Journal of Environmental Research and Public Health*, v. 15, 2018.

[3] KHAIWAL, R., SINGH, T., TRIPATHY, J. P., *et al.*, "Assessment of noise pollution in and around a sensitive zone in North India and its non-auditory impacts", *Science of the Total Environment*, v. 566-567, 2016.

[4] GUPTA, A., GUPTA, A., JAIN, K., *et al.*, "Noise Pollution and Impact on Children Health", 2018.

[5] SENZAKI, M., KADOYA, T., FRANCIS, C. D., "Direct and indirect effects of noise pollution alter biological communities in and near noise-exposed environments", *Proceedings of the Royal Society B: Biological Sciences*, v. 287, 2020.

[6] OGUNTUNDE, P. E., OKAGBUE, H. I., OGUNTUNDE, O. A., *et al.*, "A study of noise pollution measurements and possible effects on public health in ota metropolis, Nigeria", *Open Access Macedonian Journal of Medical Sciences*, v. 7, 2019.

[7] MORILLAS, J. M. B., GOZALO, G. R., GONZáLEZ, D. M., *et al.*, "Noise Pollution and Urban Planning", 2018.

[8] LIMA, T. S., BARBOSA, G. S., "*Análise Multicritério e Geoprocessamento: Identificação De Áreas Adequadas Para Implantação De Habitações Sociais Na*

Cidade Do Rio De Janeiro", 2020, Monografia (Bacharel em Engenharia Civil), UFRJ (Universidade Federal do Rio de Janeiro), Rio de Janeiro, Brasil.

[9] MAZZA, L. O., GOMES, J. G. R. C., TORRES, J. C. B., "Noise intensity prediction from video frames using deep convolutional neural networks", *Anais 23rd International Congress on Acoustics, Aachen, Alemanha*, pp. 4999–5006, 9 2019.

[10] WILDMAN, W. J., SOSIS, R., MCNAMARA, P., *Theoretical Neuroscience*, v. 4, 2014.

[11] "Neural Networks for Machine Learning", `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, Accessed: 2021-01-11.

[12] SIMONYAN, K., ZISSERMAN, A., "Very deep convolutional networks for large-scale image recognition", *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[13] RUSSAKOVSKY, O., DENG, J., SU, H., *et al.*, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision*, v. 115, n. 3, pp. 211–252, 2015.

[14] HE, K., ZHANG, X., REN, S., *et al.*, "Deep residual learning for image recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, v. 2016-December, pp. 770–778, 2016.

[15] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, v. 60, n. 6, pp. 84–90, jun 2017.

[16] SZEGEDY, C., VANHOUCKE, V., IOFFE, S., *et al.*, "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, v. 2016-December, pp. 2818–2826, 2016.

[17] WANG, M., LIU, B., FOROOSH, H., "Factorized Convolutional Neural Networks". In: *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*, v. 2018-January, 2017.

[18] JADERBERG, M., VEDALDI, A., ZISSERMAN, A., "Speeding up convolutional neural networks with low rank expansions". In: *BMVC 2014 - Proceedings of the British Machine Vision Conference 2014*, 2014.

[19] BENGIO, Y., SIMARD, P., FRASCONI, P., "Learning Long-Term Dependencies with Gradient Descent is Difficult", *IEEE Transactions on Neural Networks*, v. 5, n. 2, pp. 157–166, 1994.

[20] HOCHREITER, S., SCHMIDHUBER, J., "Long Short-Term Memory", *Neural Computation*, v. 9, n. 8, pp. 1735–1780, 1997.

[21] CARREIRA, J., ZISSERMAN, A., "Quo Vadis, action recognition? A new model and the kinetics dataset", *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, v. 2017-Janua, pp. 4724–4733, 2017.

[22] LIN, T. Y., MAIRE, M., BELONGIE, S., *et al.*, "Microsoft COCO: Common objects in context". v. 8693 LNCS, pp. 740–755, 2014.

[23] LIN, M., CHEN, Q., YAN, S., "Network in network", 2014.

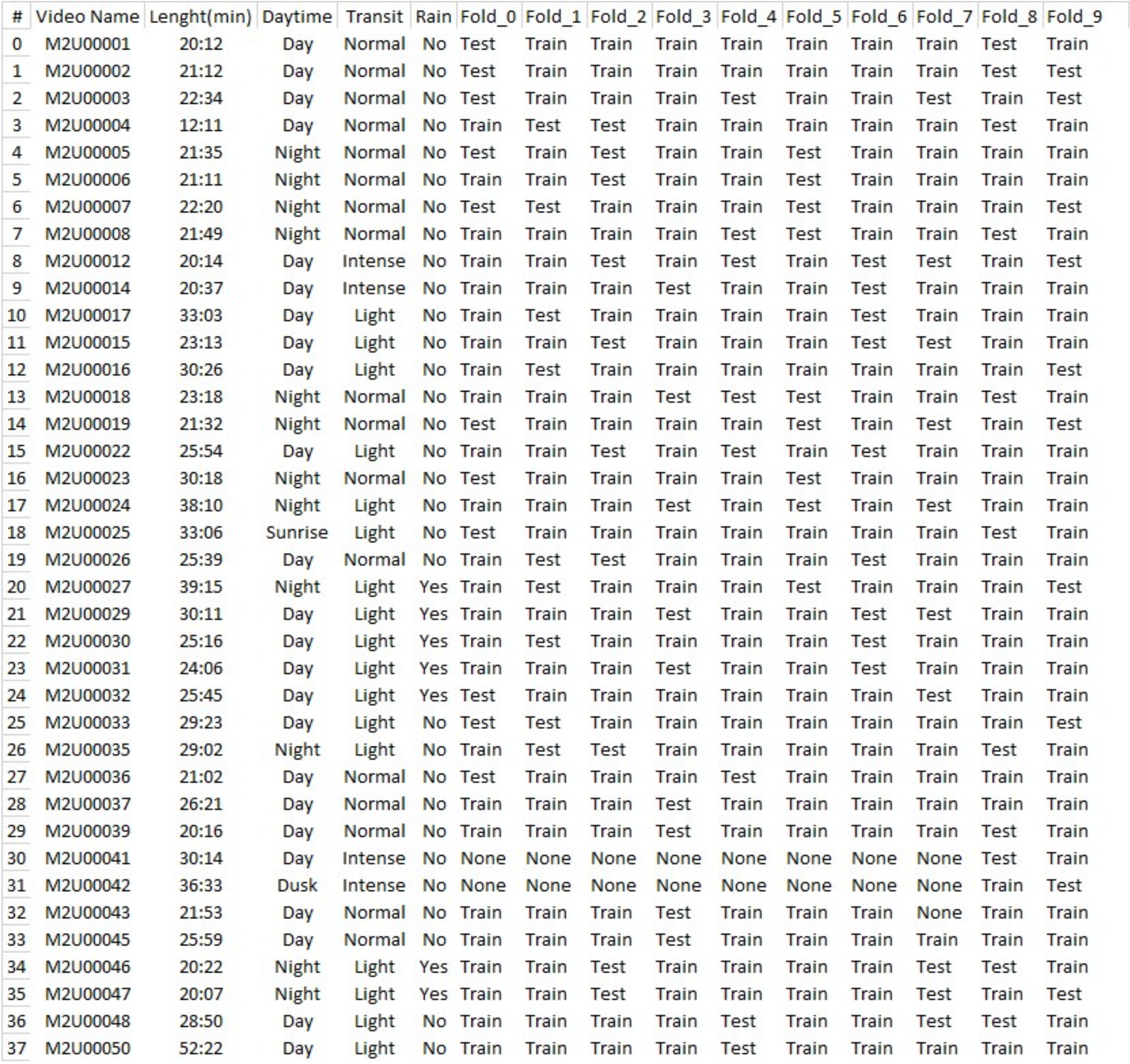
[24] TRAN, D., WANG, H., TORRESANI, L., *et al.*, "A Closer Look at Spatiotemporal Convolutions for Action Recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 6450–6459, 2018.

[25] SHI, X., CHEN, Z., WANG, H., *et al.*, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting", *Advances in Neural Information Processing Systems*, v. 2015-Janua, pp. 802–810, 2015.

Appendix A

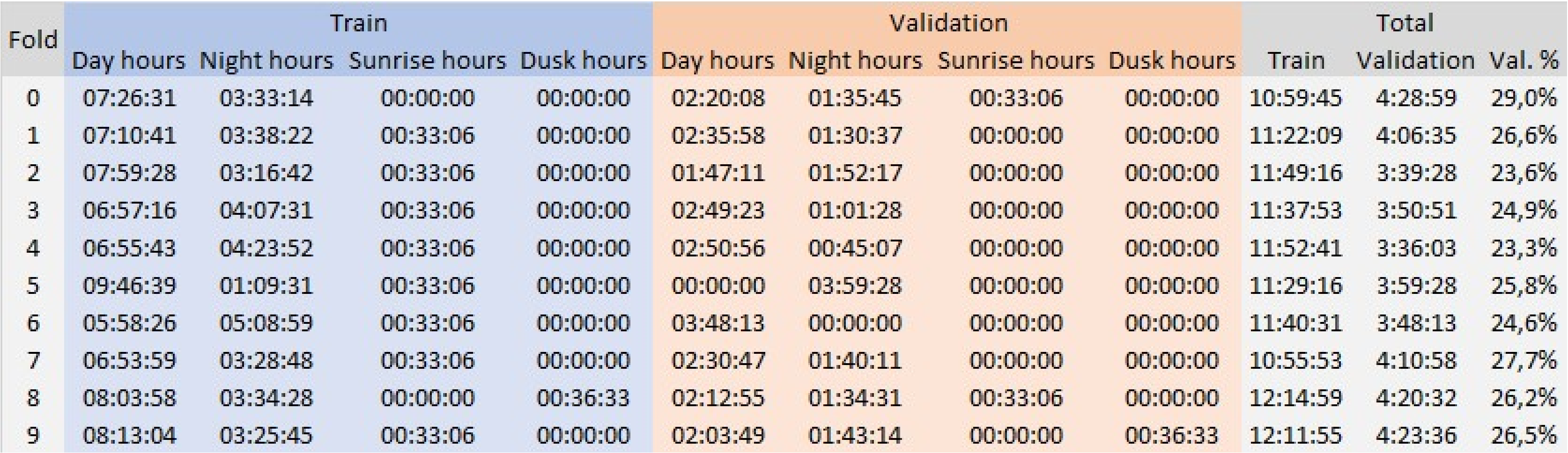
Detailed table of folds

Table A.1: Table of videos per fold



#	Video Name	Lenght(min)	Daytime	Transit	Rain	Fold_0	Fold_1	Fold_2	Fold_3	Fold_4	Fold_5	Fold_6	Fold_7	Fold_8	Fold_9
0	M2U00001	20:12	Day	Normal	No	Test	Train	Train	Train	Train	Train	Train	Train	Test	Train
1	M2U00002	21:12	Day	Normal	No	Test	Train	Train	Train	Train	Train	Train	Train	Test	Test
2	M2U00003	22:34	Day	Normal	No	Test	Train	Train	Train	Test	Train	Train	Test	Train	Test
3	M2U00004	12:11	Day	Normal	No	Train	Test	Test	Train	Train	Train	Train	Train	Test	Train
4	M2U00005	21:35	Night	Normal	No	Test	Train	Test	Train	Train	Test	Train	Train	Train	Train
5	M2U00006	21:11	Night	Normal	No	Train	Train	Test	Train	Train	Test	Train	Train	Train	Train
6	M2U00007	22:20	Night	Normal	No	Test	Test	Train	Train	Train	Test	Train	Train	Train	Test
7	M2U00008	21:49	Night	Normal	No	Train	Train	Train	Train	Test	Test	Train	Train	Test	Train
8	M2U00012	20:14	Day	Intense	No	Train	Train	Test	Train	Test	Train	Test	Test	Train	Test
9	M2U00014	20:37	Day	Intense	No	Train	Train	Train	Test	Train	Train	Test	Train	Train	Train
10	M2U00017	33:03	Day	Light	No	Train	Test	Train	Train	Train	Train	Test	Train	Train	Train
11	M2U00015	23:13	Day	Light	No	Train	Train	Test	Train	Train	Train	Test	Test	Train	Train
12	M2U00016	30:26	Day	Light	No	Train	Test	Train	Train	Train	Train	Train	Train	Train	Test
13	M2U00018	23:18	Night	Normal	No	Train	Train	Train	Test	Test	Test	Train	Train	Test	Train
14	M2U00019	21:32	Night	Normal	No	Test	Train	Train	Train	Train	Test	Train	Test	Train	Test
15	M2U00022	25:54	Day	Light	No	Train	Train	Test	Train	Test	Train	Test	Train	Train	Train
16	M2U00023	30:18	Night	Normal	No	Test	Train	Train	Train	Train	Test	Train	Train	Train	Train
17	M2U00024	38:10	Night	Light	No	Train	Train	Train	Test	Train	Test	Train	Test	Train	Train
18	M2U00025	33:06	Sunrise	Light	No	Test	Train	Train	Train	Train	Train	Train	Train	Test	Train
19	M2U00026	25:39	Day	Normal	No	Train	Test	Test	Train	Train	Train	Test	Train	Train	Train
20	M2U00027	39:15	Night	Light	Yes	Train	Test	Train	Train	Train	Test	Train	Train	Train	Test
21	M2U00029	30:11	Day	Light	Yes	Train	Train	Train	Test	Train	Train	Test	Test	Train	Train
22	M2U00030	25:16	Day	Light	Yes	Train	Test	Train	Train	Train	Train	Test	Train	Train	Train
23	M2U00031	24:06	Day	Light	Yes	Train	Train	Train	Test	Train	Train	Test	Train	Train	Train
24	M2U00032	25:45	Day	Light	Yes	Test	Train	Train	Train	Train	Train	Train	Test	Train	Train
25	M2U00033	29:23	Day	Light	No	Test	Test	Train	Train	Train	Train	Train	Train	Train	Test
26	M2U00035	29:02	Night	Light	No	Train	Test	Test	Train	Train	Train	Train	Train	Test	Train
27	M2U00036	21:02	Day	Normal	No	Test	Train	Train	Train	Test	Train	Train	Train	Train	Train
28	M2U00037	26:21	Day	Normal	No	Train	Train	Train	Test	Train	Train	Train	Train	Train	Train
29	M2U00039	20:16	Day	Normal	No	Train	Train	Train	Test	Train	Train	Train	Train	Test	Train
30	M2U00041	30:14	Day	Intense	No	None	None	None	None	None	None	None	None	Test	Train
31	M2U00042	36:33	Dusk	Intense	No	None	None	None	None	None	None	None	None	Train	Test
32	M2U00043	21:53	Day	Normal	No	Train	Train	Train	Test	Train	Train	Train	None	Train	Train
33	M2U00045	25:59	Day	Normal	No	Train	Train	Train	Test	Train	Train	Train	Train	Train	Train
34	M2U00046	20:22	Night	Light	Yes	Train	Train	Test	Train	Train	Train	Train	Test	Test	Train
35	M2U00047	20:07	Night	Light	Yes	Train	Train	Test	Train	Train	Train	Train	Test	Train	Test
36	M2U00048	28:50	Day	Light	No	Train	Train	Train	Train	Test	Train	Train	Test	Test	Train
37	M2U00050	52:22	Day	Light	No	Train	Train	Train	Train	Test	Train	Train	Train	Train	Train

Table A.2: Table of fold hours

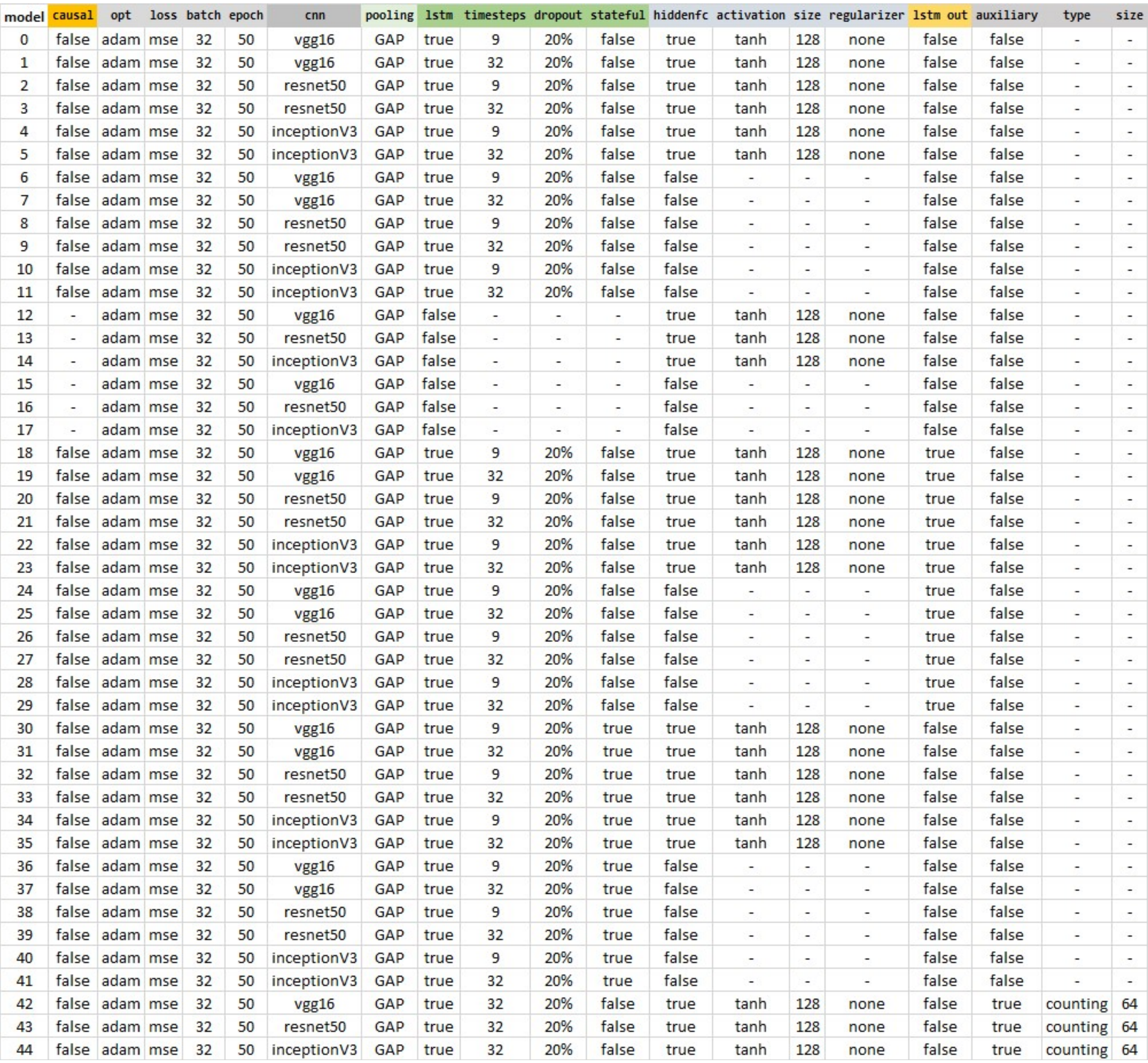


Fold	Train				Validation				Total		
	Day hours	Night hours	Sunrise hours	Dusk hours	Day hours	Night hours	Sunrise hours	Dusk hours	Train	Validation	Val. %
0	07:26:31	03:33:14	00:00:00	00:00:00	02:20:08	01:35:45	00:33:06	00:00:00	10:59:45	4:28:59	29,0%
1	07:10:41	03:38:22	00:33:06	00:00:00	02:35:58	01:30:37	00:00:00	00:00:00	11:22:09	4:06:35	26,6%
2	07:59:28	03:16:42	00:33:06	00:00:00	01:47:11	01:52:17	00:00:00	00:00:00	11:49:16	3:39:28	23,6%
3	06:57:16	04:07:31	00:33:06	00:00:00	02:49:23	01:01:28	00:00:00	00:00:00	11:37:53	3:50:51	24,9%
4	06:55:43	04:23:52	00:33:06	00:00:00	02:50:56	00:45:07	00:00:00	00:00:00	11:52:41	3:36:03	23,3%
5	09:46:39	01:09:31	00:33:06	00:00:00	00:00:00	03:59:28	00:00:00	00:00:00	11:29:16	3:59:28	25,8%
6	05:58:26	05:08:59	00:33:06	00:00:00	03:48:13	00:00:00	00:00:00	00:00:00	11:40:31	3:48:13	24,6%
7	06:53:59	03:28:48	00:33:06	00:00:00	02:30:47	01:40:11	00:00:00	00:00:00	10:55:53	4:10:58	27,7%
8	08:03:58	03:34:28	00:00:00	00:36:33	02:12:55	01:34:31	00:33:06	00:00:00	12:14:59	4:20:32	26,2%
9	08:13:04	03:25:45	00:33:06	00:00:00	02:03:49	01:43:14	00:00:00	00:36:33	12:11:55	4:23:36	26,5%

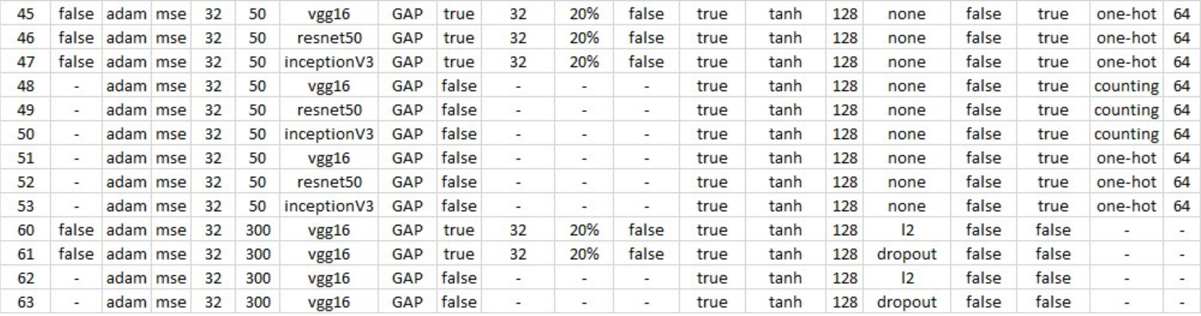
Appendix B

Detailed table of trained networks

Table B.1: Table of trained networks in multiple folds

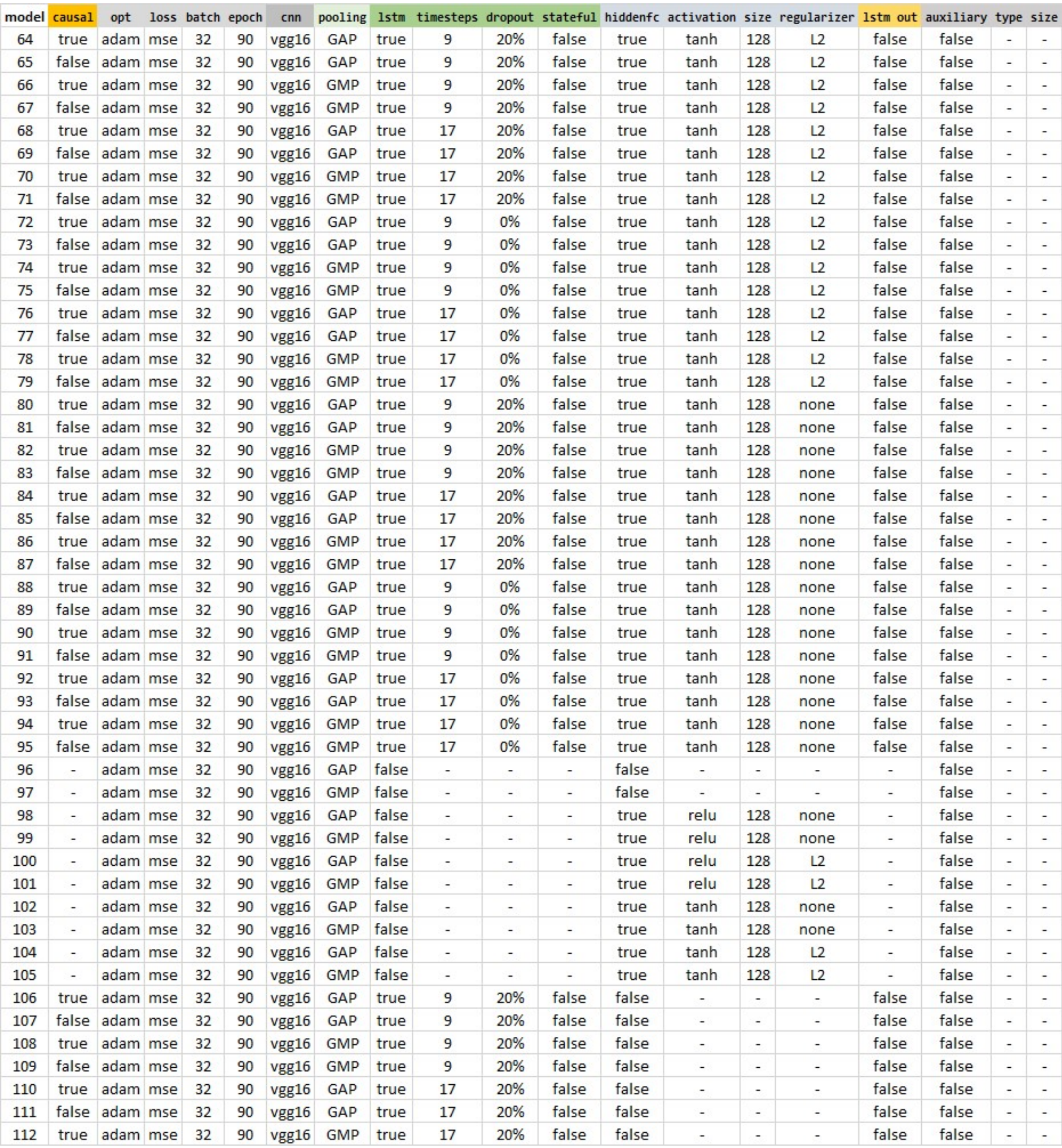


model	causal	opt	loss	batch	epoch	cnn	pooling	lstm	timesteps	dropout	stateful	hiddenfc	activation	size	regularizer	lstm out	auxiliary	type	size
0	false	adam	mse	32	50	vgg16	GAP	true	9	20%	false	true	tanh	128	none	false	false	-	-
1	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	true	tanh	128	none	false	false	-	-
2	false	adam	mse	32	50	resnet50	GAP	true	9	20%	false	true	tanh	128	none	false	false	-	-
3	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	true	tanh	128	none	false	false	-	-
4	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	false	true	tanh	128	none	false	false	-	-
5	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	true	tanh	128	none	false	false	-	-
6	false	adam	mse	32	50	vgg16	GAP	true	9	20%	false	false	-	-	-	false	false	-	-
7	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	false	-	-	-	false	false	-	-
8	false	adam	mse	32	50	resnet50	GAP	true	9	20%	false	false	-	-	-	false	false	-	-
9	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	false	-	-	-	false	false	-	-
10	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	false	false	-	-	-	false	false	-	-
11	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	false	-	-	-	false	false	-	-
12	-	adam	mse	32	50	vgg16	GAP	false	-	-	-	true	tanh	128	none	false	false	-	-
13	-	adam	mse	32	50	resnet50	GAP	false	-	-	-	true	tanh	128	none	false	false	-	-
14	-	adam	mse	32	50	inceptionV3	GAP	false	-	-	-	true	tanh	128	none	false	false	-	-
15	-	adam	mse	32	50	vgg16	GAP	false	-	-	-	false	-	-	-	false	false	-	-
16	-	adam	mse	32	50	resnet50	GAP	false	-	-	-	false	-	-	-	false	false	-	-
17	-	adam	mse	32	50	inceptionV3	GAP	false	-	-	-	false	-	-	-	false	false	-	-
18	false	adam	mse	32	50	vgg16	GAP	true	9	20%	false	true	tanh	128	none	true	false	-	-
19	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	true	tanh	128	none	true	false	-	-
20	false	adam	mse	32	50	resnet50	GAP	true	9	20%	false	true	tanh	128	none	true	false	-	-
21	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	true	tanh	128	none	true	false	-	-
22	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	false	true	tanh	128	none	true	false	-	-
23	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	true	tanh	128	none	true	false	-	-
24	false	adam	mse	32	50	vgg16	GAP	true	9	20%	false	false	-	-	-	true	false	-	-
25	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	false	-	-	-	true	false	-	-
26	false	adam	mse	32	50	resnet50	GAP	true	9	20%	false	false	-	-	-	true	false	-	-
27	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	false	-	-	-	true	false	-	-
28	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	false	false	-	-	-	true	false	-	-
29	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	false	-	-	-	true	false	-	-
30	false	adam	mse	32	50	vgg16	GAP	true	9	20%	true	true	tanh	128	none	false	false	-	-
31	false	adam	mse	32	50	vgg16	GAP	true	32	20%	true	true	tanh	128	none	false	false	-	-
32	false	adam	mse	32	50	resnet50	GAP	true	9	20%	true	true	tanh	128	none	false	false	-	-
33	false	adam	mse	32	50	resnet50	GAP	true	32	20%	true	true	tanh	128	none	false	false	-	-
34	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	true	true	tanh	128	none	false	false	-	-
35	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	true	true	tanh	128	none	false	false	-	-
36	false	adam	mse	32	50	vgg16	GAP	true	9	20%	true	false	-	-	-	false	false	-	-
37	false	adam	mse	32	50	vgg16	GAP	true	32	20%	true	false	-	-	-	false	false	-	-
38	false	adam	mse	32	50	resnet50	GAP	true	9	20%	true	false	-	-	-	false	false	-	-
39	false	adam	mse	32	50	resnet50	GAP	true	32	20%	true	false	-	-	-	false	false	-	-
40	false	adam	mse	32	50	inceptionV3	GAP	true	9	20%	true	false	-	-	-	false	false	-	-
41	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	true	false	-	-	-	false	false	-	-
42	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	true	tanh	128	none	false	true	counting	64
43	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	true	tanh	128	none	false	true	counting	64
44	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	true	tanh	128	none	false	true	counting	64

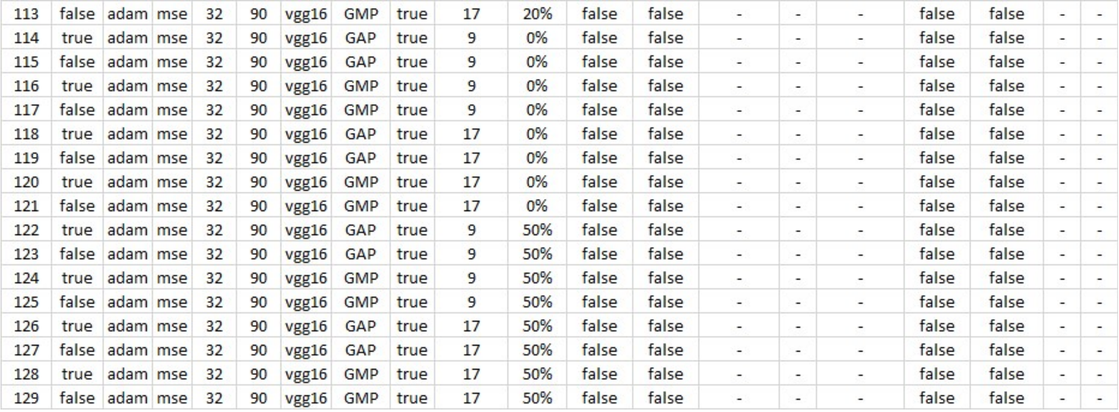


45	false	adam	mse	32	50	vgg16	GAP	true	32	20%	false	true	tanh	128	none	false	true	one-hot	64
46	false	adam	mse	32	50	resnet50	GAP	true	32	20%	false	true	tanh	128	none	false	true	one-hot	64
47	false	adam	mse	32	50	inceptionV3	GAP	true	32	20%	false	true	tanh	128	none	false	true	one-hot	64
48	-	adam	mse	32	50	vgg16	GAP	false	-	-	-	true	tanh	128	none	false	true	counting	64
49	-	adam	mse	32	50	resnet50	GAP	false	-	-	-	true	tanh	128	none	false	true	counting	64
50	-	adam	mse	32	50	inceptionV3	GAP	false	-	-	-	true	tanh	128	none	false	true	counting	64
51	-	adam	mse	32	50	vgg16	GAP	false	-	-	-	true	tanh	128	none	false	true	one-hot	64
52	-	adam	mse	32	50	resnet50	GAP	false	-	-	-	true	tanh	128	none	false	true	one-hot	64
53	-	adam	mse	32	50	inceptionV3	GAP	false	-	-	-	true	tanh	128	none	false	true	one-hot	64
60	false	adam	mse	32	300	vgg16	GAP	true	32	20%	false	true	tanh	128	l2	false	false	-	-
61	false	adam	mse	32	300	vgg16	GAP	true	32	20%	false	true	tanh	128	dropout	false	false	-	-
62	-	adam	mse	32	300	vgg16	GAP	false	-	-	-	true	tanh	128	l2	false	false	-	-
63	-	adam	mse	32	300	vgg16	GAP	false	-	-	-	true	tanh	128	dropout	false	false	-	-

Table B.2: Table of trained networks in one fold



model	causal	opt	loss	batch	epoch	cnn	pooling	lstm	timesteps	dropout	stateful	hiddenfc	activation	size	regularizer	lstm out	auxiliary	type	size
64	true	adam	mse	32	90	vgg16	GAP	true	9	20%	false	true	tanh	128	L2	false	false	-	-
65	false	adam	mse	32	90	vgg16	GAP	true	9	20%	false	true	tanh	128	L2	false	false	-	-
66	true	adam	mse	32	90	vgg16	GMP	true	9	20%	false	true	tanh	128	L2	false	false	-	-
67	false	adam	mse	32	90	vgg16	GMP	true	9	20%	false	true	tanh	128	L2	false	false	-	-
68	true	adam	mse	32	90	vgg16	GAP	true	17	20%	false	true	tanh	128	L2	false	false	-	-
69	false	adam	mse	32	90	vgg16	GAP	true	17	20%	false	true	tanh	128	L2	false	false	-	-
70	true	adam	mse	32	90	vgg16	GMP	true	17	20%	false	true	tanh	128	L2	false	false	-	-
71	false	adam	mse	32	90	vgg16	GMP	true	17	20%	false	true	tanh	128	L2	false	false	-	-
72	true	adam	mse	32	90	vgg16	GAP	true	9	0%	false	true	tanh	128	L2	false	false	-	-
73	false	adam	mse	32	90	vgg16	GAP	true	9	0%	false	true	tanh	128	L2	false	false	-	-
74	true	adam	mse	32	90	vgg16	GMP	true	9	0%	false	true	tanh	128	L2	false	false	-	-
75	false	adam	mse	32	90	vgg16	GMP	true	9	0%	false	true	tanh	128	L2	false	false	-	-
76	true	adam	mse	32	90	vgg16	GAP	true	17	0%	false	true	tanh	128	L2	false	false	-	-
77	false	adam	mse	32	90	vgg16	GAP	true	17	0%	false	true	tanh	128	L2	false	false	-	-
78	true	adam	mse	32	90	vgg16	GMP	true	17	0%	false	true	tanh	128	L2	false	false	-	-
79	false	adam	mse	32	90	vgg16	GMP	true	17	0%	false	true	tanh	128	L2	false	false	-	-
80	true	adam	mse	32	90	vgg16	GAP	true	9	20%	false	true	tanh	128	none	false	false	-	-
81	false	adam	mse	32	90	vgg16	GAP	true	9	20%	false	true	tanh	128	none	false	false	-	-
82	true	adam	mse	32	90	vgg16	GMP	true	9	20%	false	true	tanh	128	none	false	false	-	-
83	false	adam	mse	32	90	vgg16	GMP	true	9	20%	false	true	tanh	128	none	false	false	-	-
84	true	adam	mse	32	90	vgg16	GAP	true	17	20%	false	true	tanh	128	none	false	false	-	-
85	false	adam	mse	32	90	vgg16	GAP	true	17	20%	false	true	tanh	128	none	false	false	-	-
86	true	adam	mse	32	90	vgg16	GMP	true	17	20%	false	true	tanh	128	none	false	false	-	-
87	false	adam	mse	32	90	vgg16	GMP	true	17	20%	false	true	tanh	128	none	false	false	-	-
88	true	adam	mse	32	90	vgg16	GAP	true	9	0%	false	true	tanh	128	none	false	false	-	-
89	false	adam	mse	32	90	vgg16	GAP	true	9	0%	false	true	tanh	128	none	false	false	-	-
90	true	adam	mse	32	90	vgg16	GMP	true	9	0%	false	true	tanh	128	none	false	false	-	-
91	false	adam	mse	32	90	vgg16	GMP	true	9	0%	false	true	tanh	128	none	false	false	-	-
92	true	adam	mse	32	90	vgg16	GAP	true	17	0%	false	true	tanh	128	none	false	false	-	-
93	false	adam	mse	32	90	vgg16	GAP	true	17	0%	false	true	tanh	128	none	false	false	-	-
94	true	adam	mse	32	90	vgg16	GMP	true	17	0%	false	true	tanh	128	none	false	false	-	-
95	false	adam	mse	32	90	vgg16	GMP	true	17	0%	false	true	tanh	128	none	false	false	-	-
96	-	adam	mse	32	90	vgg16	GAP	false	-	-	-	false	-	-	-	-	false	-	-
97	-	adam	mse	32	90	vgg16	GMP	false	-	-	-	false	-	-	-	-	false	-	-
98	-	adam	mse	32	90	vgg16	GAP	false	-	-	-	true	relu	128	none	-	false	-	-
99	-	adam	mse	32	90	vgg16	GMP	false	-	-	-	true	relu	128	none	-	false	-	-
100	-	adam	mse	32	90	vgg16	GAP	false	-	-	-	true	relu	128	L2	-	false	-	-
101	-	adam	mse	32	90	vgg16	GMP	false	-	-	-	true	relu	128	L2	-	false	-	-
102	-	adam	mse	32	90	vgg16	GAP	false	-	-	-	true	tanh	128	none	-	false	-	-
103	-	adam	mse	32	90	vgg16	GMP	false	-	-	-	true	tanh	128	none	-	false	-	-
104	-	adam	mse	32	90	vgg16	GAP	false	-	-	-	true	tanh	128	L2	-	false	-	-
105	-	adam	mse	32	90	vgg16	GMP	false	-	-	-	true	tanh	128	L2	-	false	-	-
106	true	adam	mse	32	90	vgg16	GAP	true	9	20%	false	false	-	-	-	false	false	-	-
107	false	adam	mse	32	90	vgg16	GAP	true	9	20%	false	false	-	-	-	false	false	-	-
108	true	adam	mse	32	90	vgg16	GMP	true	9	20%	false	false	-	-	-	false	false	-	-
109	false	adam	mse	32	90	vgg16	GMP	true	9	20%	false	false	-	-	-	false	false	-	-
110	true	adam	mse	32	90	vgg16	GAP	true	17	20%	false	false	-	-	-	false	false	-	-
111	false	adam	mse	32	90	vgg16	GAP	true	17	20%	false	false	-	-	-	false	false	-	-
112	true	adam	mse	32	90	vgg16	GMP	true	17	20%	false	false	-	-	-	false	false	-	-



113	false	adam	mse	32	90	vgg16	GMP	true	17	20%	false	false	-	-	-	false	false	-	-
114	true	adam	mse	32	90	vgg16	GAP	true	9	0%	false	false	-	-	-	false	false	-	-
115	false	adam	mse	32	90	vgg16	GAP	true	9	0%	false	false	-	-	-	false	false	-	-
116	true	adam	mse	32	90	vgg16	GMP	true	9	0%	false	false	-	-	-	false	false	-	-
117	false	adam	mse	32	90	vgg16	GMP	true	9	0%	false	false	-	-	-	false	false	-	-
118	true	adam	mse	32	90	vgg16	GAP	true	17	0%	false	false	-	-	-	false	false	-	-
119	false	adam	mse	32	90	vgg16	GAP	true	17	0%	false	false	-	-	-	false	false	-	-
120	true	adam	mse	32	90	vgg16	GMP	true	17	0%	false	false	-	-	-	false	false	-	-
121	false	adam	mse	32	90	vgg16	GMP	true	17	0%	false	false	-	-	-	false	false	-	-
122	true	adam	mse	32	90	vgg16	GAP	true	9	50%	false	false	-	-	-	false	false	-	-
123	false	adam	mse	32	90	vgg16	GAP	true	9	50%	false	false	-	-	-	false	false	-	-
124	true	adam	mse	32	90	vgg16	GMP	true	9	50%	false	false	-	-	-	false	false	-	-
125	false	adam	mse	32	90	vgg16	GMP	true	9	50%	false	false	-	-	-	false	false	-	-
126	true	adam	mse	32	90	vgg16	GAP	true	17	50%	false	false	-	-	-	false	false	-	-
127	false	adam	mse	32	90	vgg16	GAP	true	17	50%	false	false	-	-	-	false	false	-	-
128	true	adam	mse	32	90	vgg16	GMP	true	17	50%	false	false	-	-	-	false	false	-	-
129	false	adam	mse	32	90	vgg16	GMP	true	17	50%	false	false	-	-	-	false	false	-	-