



SOUND PRESSURE LEVEL PREDICTION FROM VIDEO FRAMES USING
DEEP CONVOLUTIONAL NEURAL NETWORKS

Leonardo Oliveira Mazza

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: José Gabriel Rodriguez Carneiro
Gomes

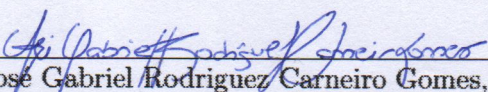
Rio de Janeiro
Junho de 2019

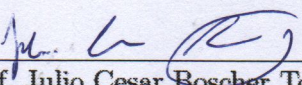
SOUND PRESSURE LEVEL PREDICTION FROM VIDEO FRAMES USING
DEEP CONVOLUTIONAL NEURAL NETWORKS

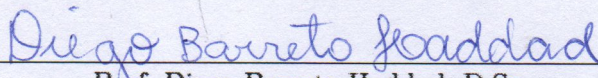
Leonardo Oliveira Mazza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO
ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE
ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE
JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A
OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA
ELÉTRICA.

Examinada por:


Prof. Jose Gabriel Rodriguez Carneiro Gomes, Ph.D.


Prof. Julio Cesar Boscher Torres, D.Sc.


Prof. Diego Barreto Haddad, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2019

Mazza, Leonardo Oliveira

Sound pressure level prediction from video frames using deep convolutional neural networks/Leonardo Oliveira Mazza. – Rio de Janeiro: UFRJ/COPPE, 2019.

XIV, 58 p.: il.; 29, 7cm.

Orientador: José Gabriel Rodriguez Carneiro Gomes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 48 – 50.

1. Convolutional neural networks. 2. Traffic noise intensity. 3. Non-linear regression. 4. Non-linear prediction. I. Gomes, José Gabriel Rodriguez Carneiro. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

*Às minhas priminhas,
Rafa e Duda*

Agradecimentos

Agradeço à minha família por possibilitar que eu pudesse seguir nessa empreitada acadêmica.

Agradeço à Tatiane Tuha pelo carinho e pelo apoio.

Agradeço ao meu orientador José Gabriel por confiar em mim ao me incluir em diversos projetos interessantes e divertidos. Agradeço também pelas várias discussões que tanto me agregaram.

Agradeço a todos os colegas do PADS pelo excelente ambiente de trabalho. Em especial, agradeço ao Olavo, Cayres, Estêvão, Bandeira, Ítalo, Fernanda, Gustavo e Renan.

Agradeço ao Pedro Savarese pela base em redes neurais e pelas consultas ao longo do trabalho.

Agradeço à CAPES pelo apoio financeiro.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

PREDIÇÃO DO NÍVEL DE PRESSÃO SONORA A PARTIR DE FRAMES DE VÍDEO COM REDES NEURAIAS CONVOLUCIONAIS PROFUNDAS

Leonardo Oliveira Mazza

Junho/2019

Orientador: José Gabriel Rodriguez Carneiro Gomes

Programa: Engenharia Elétrica

Alguns sistemas de CCTV não possuem microfones. Como resultado, a informação de pressão sonora não está disponível nesses sistemas. Um método para gerar estimativas da pressão sonora usando apenas quadros de vídeos é apresentado. Para tal, 64 combinações de modelos baseados em redes convolucionais foram treinadas a partir de uma base de dados gerada automaticamente por dados de uma câmera com um microfone mono apontada para um cruzamento com tráfego intenso de carros, caminhões e motos. Para treinar as redes neurais, imagens coloridas são usadas como entradas da rede e valores reais de pressão sonora são usados como alvos da rede. Correlação 0.607 em resultados iniciais sugere que usar valores de pressão sonora média como alvos são suficientes para que redes neurais convolucionais detectem as fontes geradoras do áudio numa cena de tráfego. Essa hipótese é testada ao se avaliar os mapas de ativação de classe (CAM) de um modelo com o formato *global average pooling*+camada *fully connected*. Por fim, os CAMs ressaltaram fortemente objetos associados a altos valores de pressão sonora como ônibus e realçaram fracamente objetos associados a menores níveis de pressão sonora como carros. Foi feita validação cruzada no modelo com menor MSE com 6 *folds* e melhor modelo foi avaliado no conjunto de teste. Esse modelo obteve correlação próxima de 0.6 em três dos vídeos de teste e correlação 0.272 e 0.207 em outros dois vídeos de teste. A baixa correlação foi associada ao barulho constante do apito de um guarda de trânsito presente somente nesses dois últimos vídeos: característica ausente no conjunto de treino. A correlação nos dados de teste calculada conjuntamente foi de 0.647. Uma correlação de 0.844 ao usar L_{eq} com intervalo de tempo maior (1 minuto) usando todos os vídeos de teste indica que a estimação de pressão sonora de mais longo prazo é menos sensível a ruído no dataset.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

SOUND PRESSURE LEVEL PREDICTION FROM VIDEO FRAMES USING DEEP CONVOLUTIONAL NEURAL NETWORKS

Leonardo Oliveira Mazza

June/2019

Advisor: José Gabriel Rodriguez Carneiro Gomes

Department: Electrical Engineering

Some CCTV systems do not have microphones. As a result, sound pressure information is not available in such systems. A method to generate traffic sound pressure estimates using solely video frames as input data is presented. To that end, we trained several combinations of models based on pretrained convolutional networks using a dataset that was automatically generated by a single camera with a mono microphone pointing at a busy traffic crossroad with cars, trucks, and motor-bikes. For neural network training from that dataset, color images are used as neural network inputs, and true sound pressure level values are used as neural network targets. A correlation of 0.607 in preliminary results suggest that sound pressure level targets are sufficient for convolutional neural networks to detect sound generating sources within a traffic scene. This hypothesis is tested by evaluating the class activation maps (CAM) of a model with the required global average pooling+fully connected layer structure. We find that the CAM strongly highlights sources that produce large sound pressure values such as buses and faintly highlights objects associated with lower sound pressure such as cars. The neural network with the lowest MSE was cross-validated with 6 folds and the best model was evaluated in the test set. The best model attained a correlation of approximately 0.6 in three of the test videos and correlations of 0.272 and 0.207 in two of the test videos. The low correlation in the two last videos was associated with a traffic warden that constantly whistles: a characteristic not present in the training set. The overall correlation using the whole test set was 0.647. A correlation of 0.844 with a longer term (1 minute) sound pressure level (L_{eq}) estimate using all test videos indicate that estimation of longer term sound pressure levels is less sensitive to sporadic noise in the dataset.

Contents

List of Figures	x
List of Tables	xiv
1 Introduction	1
1.1 Objectives	2
1.2 Text Structure	2
2 Theory	4
2.1 Training Neural Networks	4
2.1.1 Cost Function	5
2.1.2 Optimization	6
2.2 Global Average Pooling	8
2.3 Class Activation Map	10
2.4 Transfer Learning	11
2.5 Base Models	13
2.5.1 VGG16	13
2.5.2 ResNet50	14
3 Method	19
3.1 Dataset Generation	20
3.2 Conversion from Sound Pressure to Longer L_{eq}	24
3.3 Models	26
3.4 Training	27
3.4.1 Sound Source Detection	28
3.4.2 Model Validation	29
4 Results and Discussion	31
4.1 Model with the Lowest MSE	35
4.2 Model GAP+FC with Lowest MSE	36
4.2.1 Sound Source Detection	38
4.3 Model Validation	40

5 Conclusion	45
5.1 Future Work	46
Bibliography	48
A Loss curves for the models in cross-validation	51
B Predictions of the models in cross-validation	55

List of Figures

- 2.1 A new network is created with the convolutional part of a pretrained network by removing the original FC layer and substituting it for another randomly initialized FC layer. The convolutional part of a pretrained network is used in this example as a feature extractor. Other structures are also possible: not only a new FC layer could be added after the convolutional layer, but also different combinations of GAP+FC, FC+FC or GAP+FC+FC, where FC+FC means two FC layers in tandem. 12
- 2.2 Structure of VGG16: full classification network with the FC layers. The indicated convolutional part is commonly used as feature extractor. Each convolutional layer is indicated by $\text{conv } A, B \times C$. A is the number of filters, B the height of each filter and C the width of each filter. The parentheses after max pooling indicate the size of the pooling window. FC are fully connected layers with 4096, 4096 and 1000 neurons respectively. 14
- 2.3 The building block in residual learning. The dashed block represents the function $\mathcal{F}(\mathbf{x})$. Weights may represent either FC layers or convolutional layers. The arrow from \mathbf{x} to the summation sign is an identity shortcut connection. 15

2.4	An identity block. Based on the residual learning building block (Figure 2.3), it applies three convolutions with number of filters equal to F1, F2 and F3 respectively (from top to bottom). The first and last convolutions apply convolution kernels of size 1×1 and the second one applies a convolution kernel of size 3×3 . The rightmost arrow pointing at the summation symbol represents an identity shortcut connection: the input is directly added to the output of the last batch normalization. In identity blocks, all convolutional layers perform operations with stride (1,1), i.e. the resolution of input feature maps is the same of the output feature maps. Since the input is directly added to the output, the number of filters F3 in the third convolution (that defines the number of feature maps at the output) is constrained to be the same as the number of feature maps of the identity block input.	17
2.5	A convolutional block. One difference between convolutional blocks and identity blocks is the presence of convolutional layers in the shortcut connection. This allows the number of feature maps at the output of the block to be different from the block input because the number of feature maps can be adjusted by this layer. Another difference is the possibility of reduction in the resolution of feature maps. The first convolutional layer may apply strided convolutions that reduce feature map size. The same strides have to be applied to the convolutional layer at the shortcut connection. This is done so that the shortcut connection output dimensions match the output dimensions of last convolutional layer in the block. For the same reason, the number of filters of the convolutional layer in the shortcut connection has to match the number of filters in the third convolutional layer.	18
3.1	Representation of the recording scheme. The camera is positioned 50 meters from the recorded area positioned in a building 40 meters away from the crossroad at 30 meters high.	21
3.2	Schematic representing the dataset generation. First row represents incoming frames one at a time with time evolving to the right. The second row represents the audio stream timeline. Around t of each frame F_t , Equation 3.1 is applied to all audio samples from $t - t_b$ and $t + t_f$ and that generates the values S_t in the third row. To each frame F_t one value S_t is associated.	23

3.3	Samples from the ten videos. Every set of four rows in each column are samples from a distinct video. The number in the upper left corner is the video name as described in section 3.1.	25
3.4	Models tested for prediction of sound pressure level values. The convolutional block represents either the convolutional part of VGG16 or ResNet50. In the case of VGG16, the output of the last convolutional layer lies in $\mathbb{R}^{7 \times 7 \times 512}$. In the first model (a), the FC layer with a single neuron contains $7 \times 7 \times 512 + 1 = 25089$ parameters. In the second model (b), the first FC layer contains $7 \times 7 \times 512 \times 128 + 128 = 3211392$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the third model (c), the FC layer contains $512 + 1 = 513$ parameters. In the fourth model (d), the first FC layer contains $512 \times 128 + 128 = 65664$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the case of ResNet50, the number of feature maps in the last convolutional layer quadruples, so that the last convolutional layer output size is $7 \times 7 \times 2048$. In the first model (a), the FC layer with a single neuron contains $7 \times 7 \times 2048 + 1 = 100353$ parameters. In the second model (b), the first FC layer contains $7 \times 7 \times 2048 \times 128 + 128 = 12845184$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the third model (c), the FC layer contains $2048 + 1 = 2049$ parameters. In the fourth model (d), the first FC layer contains $2048 \times 128 + 128 = 262272$ parameters and the second FC layer contains $128 + 1 = 129$ parameters.	26
4.1	Training and validation log loss curves for model 35. The log used is base 10.	35
4.2	Predictions of sound pressure level S_t for model 35 in training and validation data. Sound pressure levels S_t can be converted to sound pressure levels in dB through Equation 3.2.	36
4.3	Detail of the predictions in the validation data of model 35.	37
4.4	Training and validation log loss curves for model 27. The log used is base 10.	37
4.5	Predictions of sound pressure level S_t for model 27 in training and validation data.	38
4.6	Detail of the predictions in the validation data of model 27.	39
4.7	A particularly good result: when a noisy bus drives through the crossroad.	39
4.8	A typical example: a car drives through the crossroad with unwanted noisy spots around the image.	40

4.9	Predictions for model from fold 1 in the test set. True S_t are in black and predictions are in red. The dotted lines separate sections from each video. Videos are in order 4, 7, 8, 12 and 14.	42
4.10	The dotted lines separate sections from each video. Videos are in order 4, 7, 8, 12 and 14.	43
4.11	Relationship between the predicted L_{eq} and true L_{eq} (computed with $t_2 - t_1 = 1$ minute)	44
A.1	Training and validation log loss curves in the fold 1.	51
A.2	Training and validation log loss curves in the fold 2.	52
A.3	Training and validation log loss curves in the fold 3.	52
A.4	Training and validation log loss curves in the fold 4.	53
A.5	Training and validation log loss curves in the fold 5.	53
A.6	Training and validation log loss curves in the fold 6.	54
B.1	Predictions of sound pressure level S_t in fold 1. The validation part of this fold contains frames from videos 3 and 6.	55
B.2	Predictions of sound pressure level S_t in fold 2. The validation part of this fold contains frames from videos 2 and 6.	56
B.3	Predictions of sound pressure level S_t in fold 3. The validation part of this fold contains frames from videos 1 and 6.	56
B.4	Predictions of sound pressure level S_t in fold 4. The validation part of this fold contains frames from videos 3 and 5.	57
B.5	Predictions of sound pressure level S_t in fold 5. The validation part of this fold contains frames from videos 2 and 5.	57
B.6	Predictions of sound pressure level S_t in fold 6. The validation part of this fold contains frames from videos 1 and 5.	58

List of Tables

2.1	ResNet50 major block structure. This sequence of blocks is applied between the 3×3 max pooling layer and the GAP layer. The first conv basic block is the only conv block with stride 1×1	16
3.1	Four learning rate schedules by epoch. Each column is one schedule.	28
3.2	Table of the dataset folds. D_i is a daytime video with index i . N_i is a night-time video with index i . Each row is one fold.	30
4.1	Each row corresponds to one training setup. Column Network has the name of the base convolutional network. Column Opt has the name of the optimizer. Column GAP defines the presence or absence of GAP after the last convolutional layer. Column FC is the number of neurons in the hidden layer. The value “None” in this column specifies the absence of a hidden layer. lr_list is the schedule of learning rates with reductions in epochs 30 and 50. t_loss and v_loss are respectively the training loss and validation loss.	32
4.1	Training and validation loss for each configuration of the trained models.	33
4.1	Training and validation loss for each configuration of the trained models.	34
4.2	Correlations and MSE for the validation in each fold. D_1, D_2, D_3 are the daytime videos 1, 2 and 3 respectively. N_1 and N_2 are night-time videos 5 and 6 respectively.	41
4.3	Correlations and MSE for the model from fold 1 in the test set. Overall correlation using the whole test set was 0.647.	42
4.4	Correlations for model from fold 1 in the test set with L_{eq} with time interval equal to one minute converted. Correlation using the whole test set was 0.844.	42

Chapter 1

Introduction

Noise pollution is known to have negative non-auditory effects [1]. Annoyance, health issues and learning disabilities are associated with constant exposure to high noise intensity [1]. A study [2] conducted in Berlin with 1881 patients and 2234 controls showed 1.3 times the odds of myocardial infarction in men subject to 70 dB(A) (A-weighted dB) compared to men exposed to 60 dB(A) or less. The study also demonstrated that men that lived in the area with sound pressure level of 70 dB(A) or higher for over 10 years presented 1.8 times the probability of myocardial infarction. A review [3] of several studies concluded that environmental noise had significant effects on academic performance of kindergartners and children at early school years. Since noise has an impact over daily life, health and intellectual development, monitoring noise intensity is a sensible action for urban environment improvement.

Aircraft, traffic and industry are usual noise sources. Traffic in particular is ubiquitous and its monitoring is already in place in the form of closed-circuit television. However some legislatures, for example California law [4], prohibits audio recordings without two party consent. As a result, several closed-circuit television cameras either do not contain a microphone or have it disabled. To be able to monitor noise when microphones are not always available, we present a method to estimate sound pressure levels from video data (more specifically, still images without audio) only.

This dataset contains still images (i.e. frames) and their respective average sound pressure levels. Frames are fed to a model whose parameters are optimized to minimize error between the model outputs and true noise levels. Since convolutional neural networks (CNNs) are currently the edge-performance models in image classification, segmentation, localization and other computer vision problems [5–9], they were selected for this task. Particularly we use the convolutional part of a pretrained model as a feature extractor, and train a fully connected (FC) network on its top. After training, the model should approximate average sound pressure values from

previously unseen frame inputs. We also train a second network to predict average sound pressure, and then use a visualization technique to evaluate whether the network detects noise-generating sources.

Related works predict traffic sound pressure levels with neural networks [10, 11]. Generally, Equivalent continuous (A-weighted) sound level (L_{eq}) and 10 Percentile exceeded sound levels (L_{10}) are the characteristics calculated to qualify environmental noise. L_{eq} is the steady sound level with the same energy as the measured source with a time-varying sound level. L_{eq} is calculated in a time interval t_1 to t_2 by:

$$L_{eq} = 10 \log_{10} \left(\frac{1}{(t_2 - t_1)} \int_{t_1}^{t_2} \frac{p^2(t)}{p_{ref}^2} dt \right), \quad (1.1)$$

where p_{ref} is the reference acoustic pressure of $20 \times 10^{-5} \frac{N}{m^2}$ and $p(t)$ is the acoustic pressure at time t . L_{eq} is usually calculated within relatively long periods of time (from 15 minutes to 12 hours). The N Percentile exceeded sound level (L_N) is the sound level that is exceeded N percent of the time. For example, L_{10} is the 10 Percentile exceeded sound level and that represents sound levels that are exceeded only 10 percent of the time. Thus, L_{10} characterizes loud noises (in comparison to the overall sound levels). One study [10] uses neural networks to predict L_{eq} and L_{10} with three inputs: the traffic volume, the average vehicle speeds and the percentage of heavy vehicles (within a time interval). In [10], L_{eq} is computed over the period of 15 minutes. Another study [11] computes L_{eq} over one hour and has four neural network inputs: the traffic volume, the percentage of heavy vehicles, the vehicle average speeds and the width of the roadway. A significant difference of the work in this thesis if compared to [10, 11] is that they compute sound pressure levels within long periods of time, whereas, here, sound pressure levels are computed for relatively short periods of time (600 ms).

1.1 Objectives

This thesis has the objective of predicting the sound pressure level within a traffic scene by solely using video frames as input to a convolutional neural network. It also tests the hypothesis that the model has to associate image objects to sound pressure level to successfully track its trending values.

1.2 Text Structure

Chapter 2 explains concepts used throughout the thesis. It starts with the procedure of training neural networks by establishing a minimization/optimization problem

whose solution defines training. This chapter also explains global average pooling and its requirement for the class activation map visualization. In a sequence, the models VGG16 and ResNet50, used as base for the networks built in this work, are presented together with the transfer learning technique, which is required for adapting pretrained networks to new problems.

Chapter 3 shows how the dataset is developed as to permit the training of the convolutional neural network. Then, the set of models and the optimization settings for training are presented. After that, a method to filter the class activation map for visualization is shown. Finally, a dataset splitting scheme for cross-validation is displayed.

Chapter 4 illustrates the results for the predictions of sound pressure level values and the network focus for those predictions with class activation maps. This chapter also presents the numerical evaluation of the models in an initial validation set, in the cross-validation folds and in the final test set.

Chapter 5 finishes with conclusions and suggests possible future work for prediction of sound pressure level with video frames, and for detection of the sound sources within the image.

Appendix A contains the training and validation loss curves for the model in every fold. Appendix B includes the predictions of the trained network in every fold.

Chapter 2

Theory

Generally, neural networks are mathematical functions that are based on dot products followed by non linear functions. They commonly contain a large number of parameters that have to be selected for a particular task such as image classification, regression, segmentation, translation [12] and so forth. In this chapter, we present methods to train neural networks, a layer that reduces output dimension, a visualization technique to show the network focus, a method to reuse pretrained neural networks, and the structure of the original models on which the models built in the present work are based.

2.1 Training Neural Networks

Selecting neural network parameters is often called training. Although alternatives exist [13], the usual method to train neural networks is to define an optimization problem related to the task and associate the solution of that problem to the model parameters. For example, in linear regression a model could be described as:

$$y = \boldsymbol{\beta}^T \mathbf{x} + b, \quad (2.1)$$

where $\mathbf{x} \in \mathbb{R}^M, \boldsymbol{\beta} \in \mathbb{R}^M, b \in \mathbb{R}, y \in \mathbb{R}$. In an experiment, for each input \mathbf{x}_i one y_i value is collected. So with experimental data, the problem is to select $\boldsymbol{\beta}$ and b such that:

$$\left\{ \begin{array}{l} y_1 = \boldsymbol{\beta}^T \mathbf{x}_1 + b \\ y_2 = \boldsymbol{\beta}^T \mathbf{x}_2 + b \\ \vdots \\ y_i = \boldsymbol{\beta}^T \mathbf{x}_i + b \\ \vdots \\ y_K = \boldsymbol{\beta}^T \mathbf{x}_K + b \end{array} \right. \quad (2.2)$$

Since usually no single value for $\boldsymbol{\beta}$ and b exists so that the set of Equations (2.2) is solved exactly, a new related problem is defined: to minimize a measure of the solution error or residue $\mathbf{r} \in \mathbb{R}^K$:

$$\left\{ \begin{array}{l} r_1 = y_1 - (\boldsymbol{\beta}^T \mathbf{x}_1 + b) \\ r_2 = y_2 - (\boldsymbol{\beta}^T \mathbf{x}_2 + b) \\ \vdots \\ r_i = y_i - (\boldsymbol{\beta}^T \mathbf{x}_i + b) \\ \vdots \\ r_K = y_K - (\boldsymbol{\beta}^T \mathbf{x}_K + b) \end{array} \right. \quad (2.3)$$

A possible function for this purpose is the square of the norm 2 of \mathbf{r} divided by the number of samples K : the mean squared error (MSE). The problem of selecting parameters $\boldsymbol{\beta}$, b that best fit the model to data is associated with the minimization of a cost function J :

$$J(\boldsymbol{\beta}, b) = \frac{1}{K} \sum_{k=1}^K r_k^2 = \frac{\|\mathbf{r}\|_2^2}{K} \quad (2.4)$$

So the optimization problem that selects $\boldsymbol{\beta}$ and b is:

$$\underset{\boldsymbol{\beta}, b}{\text{minimize}} \quad J(\boldsymbol{\beta}, b) \quad (2.5)$$

2.1.1 Cost Function

More generally, a model represented by a function f with parameters $\boldsymbol{\theta}$ and input \mathbf{x} is optimized to minimize a measure l (loss function) of the similarity between $f(\mathbf{x}; \boldsymbol{\theta})$ and a target \mathbf{y} . In the example of Equation (2.4), l is the MSE. If \mathbf{x} and \mathbf{y} are random vectors, the cost function J is:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, \mathbf{y}} [l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})] \quad (2.6)$$

From Equation (2.6), for each set of parameters $\boldsymbol{\theta}$, one value of the cost function J is calculated. The expected value of the loss function is computed over all possible values of \mathbf{x} and \mathbf{y} , which corresponds to an integral of the loss function over all possible \mathbf{x} and \mathbf{y} weighted by their joint probability $P(\mathbf{x}, \mathbf{y})$. Since the distribution $P(\mathbf{x}, \mathbf{y})$ is usually not available, the expected value $\mathbb{E}_{\mathbf{x}, \mathbf{y}}$ is approximated with a dataset D of K pairs $(\mathbf{x}_k, \mathbf{y}_k)$ so that:

$$J(\boldsymbol{\theta}) \approx \frac{1}{K} \sum_{k=1}^K l(f(\mathbf{x}_k; \boldsymbol{\theta}), \mathbf{y}_k) \quad (2.7)$$

The choice of loss functions varies depending on the task. Examples of loss functions are:

- binary cross-entropy (for classification problems):

$$l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}) = \mathbf{y} \log(f(\mathbf{x}; \boldsymbol{\theta})) + (1 - \mathbf{y}) \log(1 - f(\mathbf{x}; \boldsymbol{\theta})) \quad (2.8)$$

- categorical cross-entropy (for classification problems):

$$l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}) = \sum_i \mathbf{y}[i] \log(f(\mathbf{x}; \boldsymbol{\theta})[i]) + (1 - \mathbf{y}[i]) \log(1 - f(\mathbf{x}; \boldsymbol{\theta})[i]) \quad (2.9)$$

where $\mathbf{y}[i]$ and $f(\mathbf{x}; \boldsymbol{\theta})[i]$ are the i -th elements of each vector.

- MSE (for regression problems, usually):

$$l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y}) = (f(\mathbf{x}; \boldsymbol{\theta}) - \mathbf{y})^2 \quad (2.10)$$

2.1.2 Optimization

Training a neural network is thus the same as optimizing the cost function J with a loss function l . Since several neural network models have millions of parameters [6, 14], optimizers with space complexity, i.e. memory requirements, $O(n^2)$ (e.g. Newton's method), where n the number of parameters to be optimized, are unfeasible. For this reason, methods to train neural networks are generally gradient-based $O(n)$ algorithms.

The gradient \mathbf{g} of the cost function J with respect to parameters $\boldsymbol{\theta}$ is:

$$\mathbf{g}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}, \mathbf{y}} [l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})] \quad (2.11)$$

Because the explicit computation of the the gradient of the expected value $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}, \mathbf{y}} [l(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{y})]$ is impractical (even if $P(\mathbf{x}, \mathbf{y})$ were available, which is not the case), The gradient \mathbf{g} is estimated as \mathbf{g}^* with a dataset D with K samples:

$$\begin{aligned} \mathbf{g}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{K} \nabla_{\boldsymbol{\theta}} \sum_{k=1}^K l(f(\mathbf{x}_k; \boldsymbol{\theta}), \mathbf{y}_k) = \\ &= \mathbf{g}^*(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} J^*(\boldsymbol{\theta}) = \frac{1}{K} \sum_{k=1}^K \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}_k; \boldsymbol{\theta}), \mathbf{y}_k), \end{aligned} \quad (2.12)$$

Equation (2.12) shows that the estimation $\mathbf{g}^*(\boldsymbol{\theta})$ of $\mathbf{g}(\boldsymbol{\theta})$ computes the average of the gradient of the loss for all data points in D . This equation shows how parallel the computation of estimate gradients can be: the expensive computation of $\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}_k; \boldsymbol{\theta}), \mathbf{y}_k)$ is independent for each pair $(\mathbf{x}_k, \mathbf{y}_k)$. As a consequence, the factors $\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}_k; \boldsymbol{\theta}), \mathbf{y}_k)$ could be calculated in parallel then averaged for the \mathbf{g}^* estimation.

When optimizing J^* over large datasets, it is common to sample a subset of the available pairs $(\mathbf{x}_i, \mathbf{y}_i)$ and compute an approximation \hat{J} from that set. This is the mini-batch strategy (as opposed to the batch/full batch approaches that use the entire dataset to compute J). Mini-batches turn practical the optimization of models over large datasets such as ImageNet (with over 1.5 million images) which would otherwise require enormous amounts of memory. Another advantage of mini-batches compared to the full batch is the larger number of iterations that optimization algorithms apply for roughly the same computation. In addition, small mini-batch sizes (at an extreme a mini-batch of size 1) increase the variance of the gradient estimation and, as a consequence, act as regularization [15]. Hardware is often optimized for larger batch sizes, so an usual trade-off is to select batch sizes from the range 16-256 [15]. For a mini-batch of size N ($N < K$), the estimation $\hat{\mathbf{g}}(\boldsymbol{\theta})$ of $\mathbf{g}(\boldsymbol{\theta})$ is similar to Equation (2.12):

$$\hat{\mathbf{g}}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}_n; \boldsymbol{\theta}), \mathbf{y}_n) \quad (2.13)$$

For each optimization step with a mini-batch, $\hat{\mathbf{g}}(\boldsymbol{\theta})$ is computed with N samples from the dataset. Usually, implementations randomize the dataset prior to dividing it in mini-batches and do not allow the same sample to be used more than once before all samples are used. An epoch is completed when all samples in the dataset are used in optimization steps. After that, the mini-batches are again randomly divided and the optimization continues in a new epoch. The optimization of neural network models commonly occurs over several epochs.

Typical optimization algorithms are stochastic gradient descent (SGD), SGD with momentum, RMSprop and Adam, where RMS stands for root mean square and Adam stands for adaptive moment estimation.

- SGD:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}_t) \quad (2.14)$$

Parameters are updated by subtracting the gradient estimate $\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}_t)$ scaled by α , the learning rate, from the current parameter array $\boldsymbol{\theta}_t$.

- SGD with momentum:

$$\boldsymbol{\mu}_{t+1} = \beta \boldsymbol{\mu}_t + (1 - \beta) \nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}_t) \quad (2.15)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \boldsymbol{\mu}_{t+1} \quad (2.16)$$

SGD with momentum filters the noisy gradient estimate $\nabla_{\boldsymbol{\theta}} \hat{J}(\boldsymbol{\theta}_t)$ by calcu-

lating a leaky accumulated gradient estimate $\boldsymbol{\mu}_{t+1}$ before applying it to parameters $\boldsymbol{\theta}_{t+1}$. The value $\beta \in (0, 1)$ determines how much of the accumulated gradient is weighted with the previous $\boldsymbol{\mu}_t$. In the Keras [16] implementation, the effect of parameters α and β is less intuitive:

$$\boldsymbol{\mu}_{t+1} = \beta\boldsymbol{\mu}_t - \alpha\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_t) \quad (2.17)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{\mu}_{t+1} \quad (2.18)$$

Here, $\boldsymbol{\mu}_{t+1}$ is obtained by filtering the gradient step $-\alpha\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_t)$ already with the negative sign, and $\boldsymbol{\mu}_{t+1}$ is no longer a convex combination of the accumulated gradient and the current gradient.

- RMSprop:

$$\boldsymbol{\nu}_{t+1} = \beta\boldsymbol{\nu}_t + (1 - \beta)(\nabla_{\boldsymbol{\theta}}\hat{J}) \circ (\nabla_{\boldsymbol{\theta}}\hat{J}) \quad (2.19)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_t) \circ \left((\boldsymbol{\nu}_{t+1} + \epsilon) \cdot \wedge \frac{-1}{2} \right) \quad (2.20)$$

where $(\nabla_{\boldsymbol{\theta}}\hat{J}) \circ (\nabla_{\boldsymbol{\theta}}\hat{J})$ operation represents an element-wise multiplication and the inverse square root $(\boldsymbol{\nu}_{t+1} + \epsilon) \cdot \wedge \frac{-1}{2}$ operation represents an element-wise inverse square root. The element-wise square of the gradient is filtered through $\boldsymbol{\nu}_{t+1}$ to serve as an estimate for the variance of each parameter. $\boldsymbol{\nu}_{t+1}$ is then used to normalize the gradient in the update Equation of $\boldsymbol{\theta}_{t+1}$. The small constant ϵ is used to avoid division by zero.

- Adam: Adam unites ideas from SGD with momentum and RMSprop. It filters the gradient $\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_t)$ through $\boldsymbol{\mu}_{t+1}$ and filters an element-wise variance estimate $(\nabla_{\boldsymbol{\theta}}\hat{J}) \circ (\nabla_{\boldsymbol{\theta}}\hat{J})$ through $\boldsymbol{\nu}_{t+1}$. It then updates the parameters with $\boldsymbol{\mu}_{t+1}$ normalized by the element-wise inverse square root of $\boldsymbol{\nu}_{t+1}$. Again, a small ϵ is used to avoid division by zero in the parameter update for $\boldsymbol{\theta}_{t+1}$ in Equation 2.23.

$$\boldsymbol{\mu}_{t+1} = \beta_1\boldsymbol{\mu}_t + (1 - \beta_1)\nabla_{\boldsymbol{\theta}}\hat{J}(\boldsymbol{\theta}_t) \quad (2.21)$$

$$\boldsymbol{\nu}_{t+1} = \beta_2\boldsymbol{\nu}_t + (1 - \beta_2)(\nabla_{\boldsymbol{\theta}}\hat{J}) \circ (\nabla_{\boldsymbol{\theta}}\hat{J}) \quad (2.22)$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\boldsymbol{\mu}_{t+1} \circ \left((\boldsymbol{\nu}_{t+1} + \epsilon) \cdot \wedge \frac{-1}{2} \right) \quad (2.23)$$

2.2 Global Average Pooling

Global average pooling [17] (GAP) is a layer commonly applied after the last convolutional layer. It was originally presented as a regularization method [17]. GAP has

other advantages as it reduces the overall parameter count and allows the application of visualization techniques to illustrate image regions relevant for the network prediction.

When dealing with image classification tasks, one typically flattens the output of the last convolutional layer and adds fully connected (FC) layers at the end. This flattening transforms a 3-D tensor into a 1-D vector. For example, the output of the last convolutional layer of VGG16 lies in $\mathbb{R}^{7 \times 7 \times 512}$. After flattening, the output lies in \mathbb{R}^{25088} . Then, in this network, there is a sequence of three FC layers: the first one with 4096 neurons, the second one with 4096 neurons, and the third one with 1000 neurons. The first FC layer has 25088×4096 neuron parameters and 4096 biases for a total of 102.7 million parameters. In addition, the second and third layers add another $4096 \times 4096 + 4096 \approx 16.7$ million and $4096 \times 1000 + 1000 \approx 4$ million parameters respectively. This high number of parameters is not only prone to overfitting, it is also computationally demanding. GAP reduces this cost by applying a function that reduces input tensor size before flattening the tensor. In the VGG16 example, GAP is a function $\mathbb{R}^{7 \times 7 \times 512} \rightarrow \mathbb{R}^{512}$. Generally, $\text{GAP} : \mathbb{R}^{M \times N \times K} \rightarrow \mathbb{R}^K$. The reduction in the number of parameters, for the VGG16 example provided, is discussed in the next paragraph.

In the last convolutional layers, the number of channels increases substantially e.g. from 3 channels in the input layer to 512 in the output layers in the VGG16 case. To avoid that the input to the first FC layer be $7 \times 7 \times 512$, we substitute each 7×7 channel by the average of the 49 numbers. That reduces the input to the first FC layer to 512: a 49 times reduction in the number of components if compared to a network without GAP. In general, operation GAP computes is the average along M and N for each individual K , i.e. it averages every $M \times N$ feature map and outputs a single value for each one of them. For an output $F \in \mathbb{R}^{M \times N \times K}$ of a convolutional layer with K feature maps each with resolution $M \times N$, the k -th element of the output vector $\mathbf{y} \in \mathbb{R}^K$ is:

$$y_k = \frac{\sum_m \sum_n F_{mn}^k}{MN}, \quad (2.24)$$

where F^k indicates the k -th feature map. In the VGG16 example, the application of GAP, followed by an FC layer with the same number neurons as before (4096), would lead to a first FC layer with $512 \times 4096 + 4096 \approx 2$ million parameters, a reduction in parameter count of approximately 49 times. However, some models do not even use hidden layers (i.e. FC layers that are not at the neural network output) after GAP [18]. In a classification task with 1000 classes performed by VGG16 with GAP and no hidden layers, the total parameter count of the FC layers is reduced from the original 123 million to 513 thousand.

2.3 Class Activation Map

Class activation map (CAM) is a method that allows visualization of the most relevant parts of an image, which are directly related to the classification of the image itself [19]. It does a linear combination of the feature maps at the output of the last convolutional layer to form an image that highlights regions of high importance. Its results demonstrated that a CNN trained for classification not only identifies the input image, but it also implicitly localizes the relevant object in the image.

This method depends on the structure of the network. To apply CAMs, it is required that the last convolutional layer be followed by a GAP and a single FC layer. It is also possible that functions that preserve the ordering of the elements in input vectors such as softmax, tanh or sigmoid be applied afterwards in classification tasks. CAM uses the linearity of the GAP and the dot product in the FC layer to compose a matrix such that the sum of each entry leads to the neuron output.

After the GAP, the output h of a single linear neuron with weights w_k and bias b is:

$$h = \sum_k w_k y_k + b, \quad (2.25)$$

where the k -th output of a GAP from a convolutional layer with K feature maps is:

$$y_k = \frac{\sum_m \sum_n F_{mn}^k}{MN} \quad (2.26)$$

If (2.26) is applied to Equation (2.25), we have:

$$\begin{aligned} h &= \sum_k w_k \frac{\sum_m \sum_n F_{mn}^k}{MN} + b \\ &= \sum_k \sum_m \sum_n \frac{w_k F_{mn}^k}{MN} + b \\ &= \sum_m \sum_n \sum_k \frac{w_k F_{mn}^k}{MN} + b \\ &= \frac{\sum_m \sum_n \sum_k w_k F_{mn}^k}{MN} + b \end{aligned} \quad (2.27)$$

Finally, the CAM is defined as the $M \times N$ frame:

$$\text{CAM}(m, n) = \sum_k w_k F_{mn}^k \quad (2.28)$$

From Equation (2.28) we see that CAM is a linear combination of K feature maps F^k with the neuron weights w_k as the weights of the linear combination. As a function of the CAM, the network output is:

$$h = \frac{\sum_m \sum_n \text{CAM}(m, n)}{MN} + b \quad (2.29)$$

From Equation (2.29) it is clear that, for CAMs, the following property holds: when their sum is divided by the constant MN and added to the bias b it yields the neuron output h . So entries in the CAM matrix with indices (m, n) that return large values add a direct contribution to the overall network output. In this example we assume that the last FC layer is composed by a single neuron. In classification tasks with more than one neuron at the output (more than two classes), each set of weights w of each neuron corresponds to a weighing of feature maps for a single class. Therefore, for a single input image in a classification task with multiple classes, several CAMs can be generated: one for each class.

A drawback of this method is the low resolution typical of the last feature maps in a CNN. Solutions to that range from classical upsampling methods such as bicubic interpolation to the modification of the original network. The original paper [19] proposed the removal of some of the last convolutional layers and pooling to yield higher spatial resolution. Then, a single convolutional layer was added with $1024 \ 3 \times 3$ filters followed by a GAP and softmax. This new layer was trained and the final network yielded results that were similar to the original ones.

With examples and benchmarks, it was shown that CAMs display the classified object position in regions where the CAM matrix has high values and achieves competitive results in object localization: a task for which it was not directly trained for [19]. Although not covered in here, other methods based on CAM [20] do not require the GAP+FC structure after the last convolutional layer to produce the visualization.

In conclusion, CAM is a method to visualize image regions with high relevance to the network output. This is accomplished by exploiting the linearity of the GAP+FC layers. It is a linear combination of feature maps weighted by the parameters of the FC layer. For a single input image, several CAMs can be formed: one for each element of the network output.

2.4 Transfer Learning

Transfer learning is a method to apply a model, which was previously trained for one task, to a different task. Patterns in data learned by a model can be sufficiently general so that its parameters are good initial conditions for other tasks. This method is usual when the training of the original network occurs with a much larger annotated dataset if compared to the dataset of the new task. For example, some models [6, 14, 18] are trained with the ImageNet [21] dataset: it contains over a million annotated images and 1000 classes. Often, the new dataset will

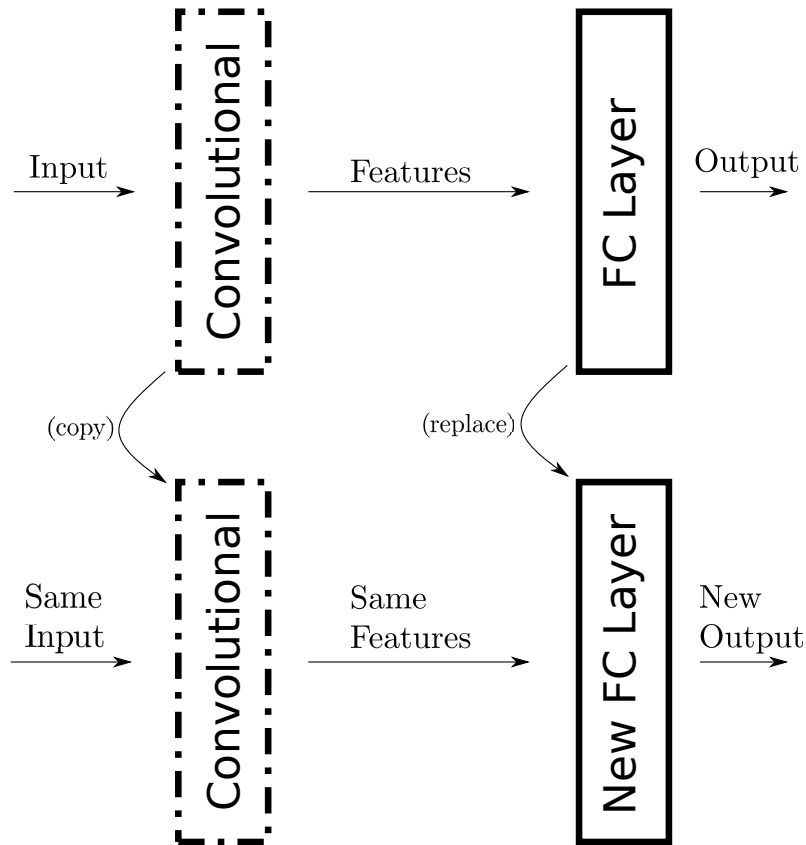


Figure 2.1: A new network is created with the convolutional part of a pretrained network by removing the original FC layer and substituting it for another randomly initialized FC layer. The convolutional part of a pretrained network is used in this example as a feature extractor. Other structures are also possible: not only a new FC layer could be added after the convolutional layer, but also different combinations of GAP+FC, FC+FC or GAP+FC+FC, where FC+FC means two FC layers in tandem.

neither be that large nor diverse. So it would be useful to reuse a model capable of retrieving features from such large dataset in a different application. In particular, classification CNNs can usually be divided into two parts: the convolutional part and the FC part. The convolutional part is often called the feature extractor as its output is related to relevant image characteristics [19]. The FC part is associated with the classification task. A typical transfer learning technique consists of removing the FC part of a pretrained CNN and using this pretrained convolutional part by attaching a new FC layer at its end. This new FC layer is randomly initialized and then either the whole network is trained in the new dataset or only the FC layer is trained. Another possibility is to add a GAP before the FC layer. This, as shown in Figure 2.2, reduces the number of new parameters to be trained. Other examples of transfer learning include models [7] that use pretrained networks as a module in whole new networks.

A great advantage of transfer learning is the ability to quickly prototype models

with low computational cost. Training a model with the ImageNet dataset requires several graphics processing units (GPUs) and days of computation unless a large processor cluster is available [22, 23]. The convolutional part of these pretrained models can be used to convert high-dimensional images with low semantic value into relatively low-dimensional vectors with high semantic value. For example, VGG16 expects an input in $\mathbb{R}^{224 \times 224 \times 3}$ and its convolutional part has an output in $\mathbb{R}^{7 \times 7 \times 512}$, which corresponds to a size reduction by a factor of six. In addition to the size reduction, if the new model only requires training of the new top FC layers, caching features speeds training up. At each parameter update, the network computes one forward pass and one backward pass per sample. Consequently, the model executes one function evaluation in the forward pass for each sample at each epoch. In the ImageNet example, that involves computing the forward pass up to the end of the convolutional part, for the entire dataset roughly one hundred times. In an example where only the FC layers are trained, it is possible to reduce the number of forward passes in the convolutional part to one. This leads to model testing at a reduced computational cost.

2.5 Base Models

The models in this work were based on two CNNs: VGG16 and ResNet50. They were chosen for their structural simplicity and high accuracy in the ImageNet challenge (ILSVRC [21]).

2.5.1 VGG16

The VGG16 neural network [14] is known for winning the localization challenge, and attaining second place in the classification challenge at the ImageNet contest of 2014. Its convolutional structure contains approximately 14 million parameters, and the fully connected (FC) top layers contain 124 million parameters. Its convolutional layers are divided into five blocks. After each block, feature maps are downsampled with a 2×2 max pooling layer. All convolutions are followed by ReLU activations. The first two FC layers (each with 4096 neurons) are also followed by ReLU activations. The final FC layer is followed by a softmax layer for classification. The last max pooling outputs a tensor in $\mathbb{R}^{7 \times 7 \times 512}$. This network achieved a top-1 classification error of 27% and top-5 error of 8.8% in the ImageNet challenge of 2014 (these classification errors are extracted from Table 3 of [14], first row of configuration D). The overall structure of VGG16 is illustrated in Figure 2.2. The convolutional part of this network serves as base (pre-trained feature extractor) for a set of models in this work.

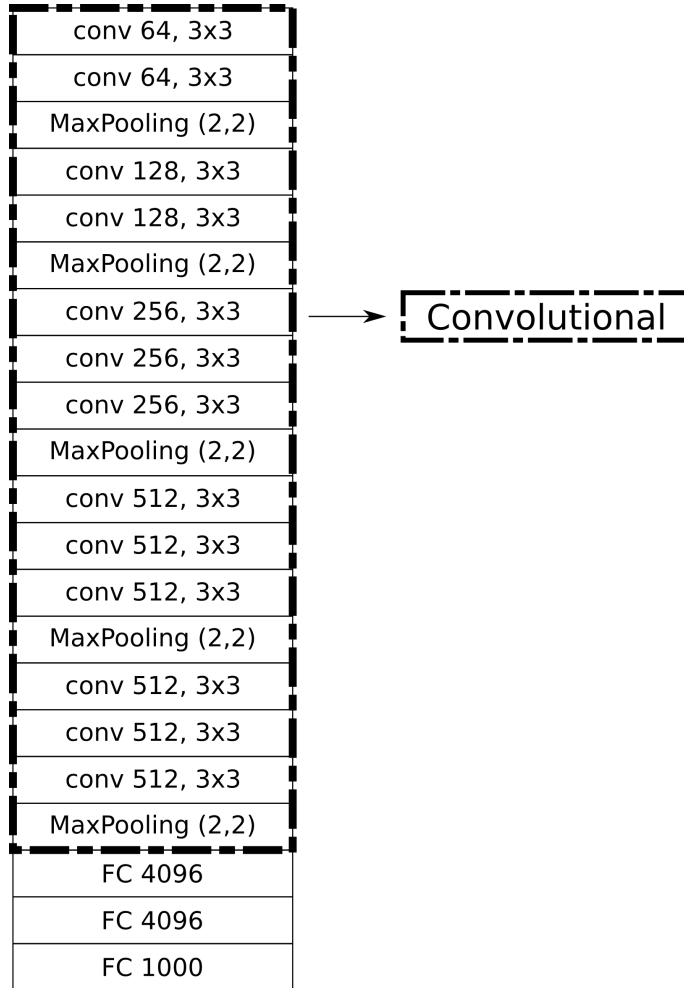


Figure 2.2: Structure of VGG16: full classification network with the FC layers. The indicated convolutional part is commonly used as feature extractor. Each convolutional layer is indicated by conv $A, B \times C$. A is the number of filters, B the height of each filter and C the width of each filter. The parentheses after max pooling indicate the size of the pooling window. FC are fully connected layers with 4096, 4096 and 1000 neurons respectively.

2.5.2 ResNet50

Training deep neural networks is challenging [6]. Gradient descent based optimizers fail to find parameters that minimize loss functions when the number of layers grows substantially. The problem of training deep networks was solved by changing the structure of the model [6]. That was done by introducing the concept of residual learning. For a network $\mathcal{H}(\mathbf{x})$, it is argued that optimizing the residue $\mathcal{H}(\mathbf{x}) - \mathbf{x}$ is an easier task than optimizing a general function. To make a function so that its parameters are optimized to learn the residual mapping $\mathcal{H}(\mathbf{x}) - \mathbf{x}$, $\mathcal{F}(\mathbf{x})$ is defined such that:

$$\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x} \tag{2.30}$$

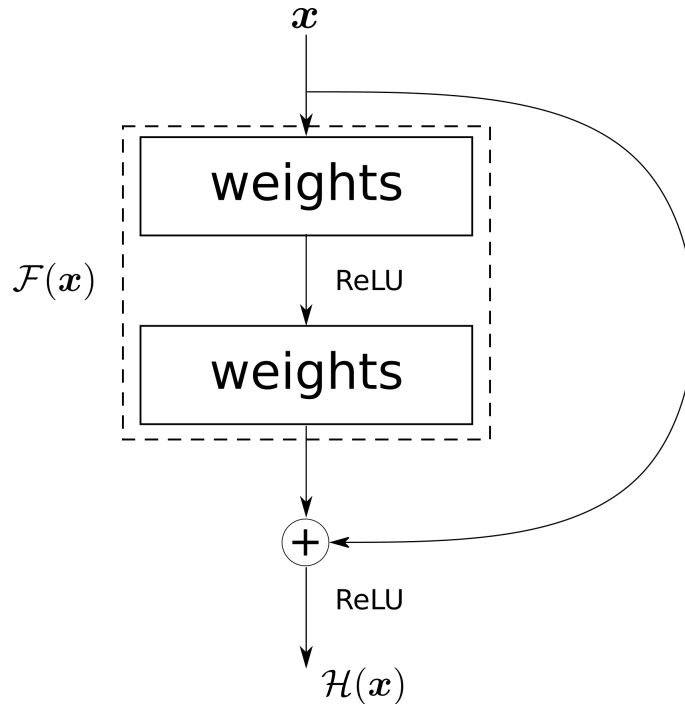


Figure 2.3: The building block in residual learning. The dashed block represents the function $\mathcal{F}(\mathbf{x})$. Weights may represent either FC layers or convolutional layers. The arrow from \mathbf{x} to the summation sign is an identity shortcut connection.

So the actual function $\mathcal{H}(\mathbf{x})$ in the network is defined as:

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x} \quad (2.31)$$

Figure 2.3 illustrates a basic block of residual learning. This layout introduces one constraint: the dimension of the input \mathbf{x} has to be the same as that of the output $\mathcal{H}(\mathbf{x})$. To overcome this issue a second type of shortcut connection is introduced: a connection with a 1×1 convolution in the shortcut. This type of shortcut can change the number of output feature maps and their resolution so that they match the output resolution and number of channels.

Based on the residual learning building block, two blocks used in ResNet50, a residual convolutional network, are created: the identity block (Figure 2.4) and the convolutional block (Figure 2.5). Identity blocks typically reduce the number of feature maps in the inner convolutions and then output the same number of feature maps as the input. Convolutional blocks may change the number of feature maps and reduce feature map spatial resolution by applying strided convolutions. The final structure of ResNet50 is similar to that of the illustrated in Figure 3 of the original paper [6] named “34-layer residual”. A difference between ResNet50 and “34-layer residual” is the presence of one extra convolutional layer in ResNet50 at each convolutional block and at each identity block. ResNet50 starts with a convolutional layer with $64 \ 7 \times 7$ filters and stride 2. After that, a 3×3 max pooling

Table 2.1: ResNet50 major block structure. This sequence of blocks is applied between the 3×3 max pooling layer and the GAP layer. The first conv basic block is the only conv block with stride 1×1 .

	Basic block	F1	F2	F3	Stride
Major block 1	conv	64	64	256	1×1
	identity	64	64	256	1×1
	identity	64	64	256	1×1
Major block 2	conv	128	128	512	2×2
	identity	128	128	512	1×1
	identity	128	128	512	1×1
	identity	128	128	512	1×1
Major block 3	conv	256	256	1024	2×2
	identity	256	256	1024	1×1
	identity	256	256	1024	1×1
	identity	256	256	1024	1×1
	identity	256	256	1024	1×1
	identity	256	256	1024	1×1
Major block 4	conv	512	512	2048	2×2
	identity	512	512	2048	1×1
	identity	512	512	2048	1×1

with stride 2 is applied. At this stage the feature map size is 56×56 . Then, a total of four major blocks, composed of basic identity and convolutional blocks, are applied. Inside each major block the first basic block is always a convolutional block and the other ones are identity blocks. The first basic block (convolutional) of the first major block is the only convolutional block in the whole network that does not reduce the input dimensions. All the other convolutional blocks cut feature map resolution by a factor of two along width and height. Inside each major block the values of F1, F2 and F3 are the same for all basic blocks. The major block structure is illustrated in Table 2.1. After the major blocks, a GAP is applied to reduce the last convolutional block output to a 2048 element vector. Finally, for the 1000-class classification task, a 1000-neuron FC layer is added.

The addition of shortcut connections allow the optimization of neural networks with more than ten times more layers than what was previously possible. Models based on this new structure attained first place in ImageNet detection, ImageNet localization, COCO [24] detection and COCO segmentation of the year 2015.

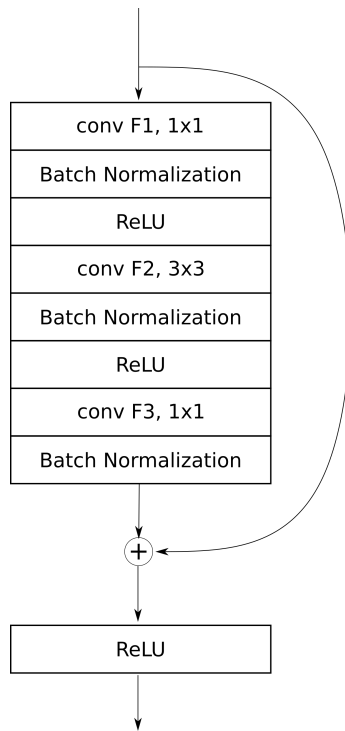


Figure 2.4: An identity block. Based on the residual learning building block (Figure 2.3), it applies three convolutions with number of filters equal to $F1$, $F2$ and $F3$ respectively (from top to bottom). The first and last convolutions apply convolution kernels of size 1×1 and the second one applies a convolution kernel of size 3×3 . The rightmost arrow pointing at the summation symbol represents an identity shortcut connection: the input is directly added to the output of the last batch normalization. In identity blocks, all convolutional layers perform operations with stride $(1,1)$, i.e. the resolution of input feature maps is the same of the output feature maps. Since the input is directly added to the output, the number of filters $F3$ in the third convolution (that defines the number of feature maps at the output) is constrained to be the same as the number of feature maps of the identity block input.

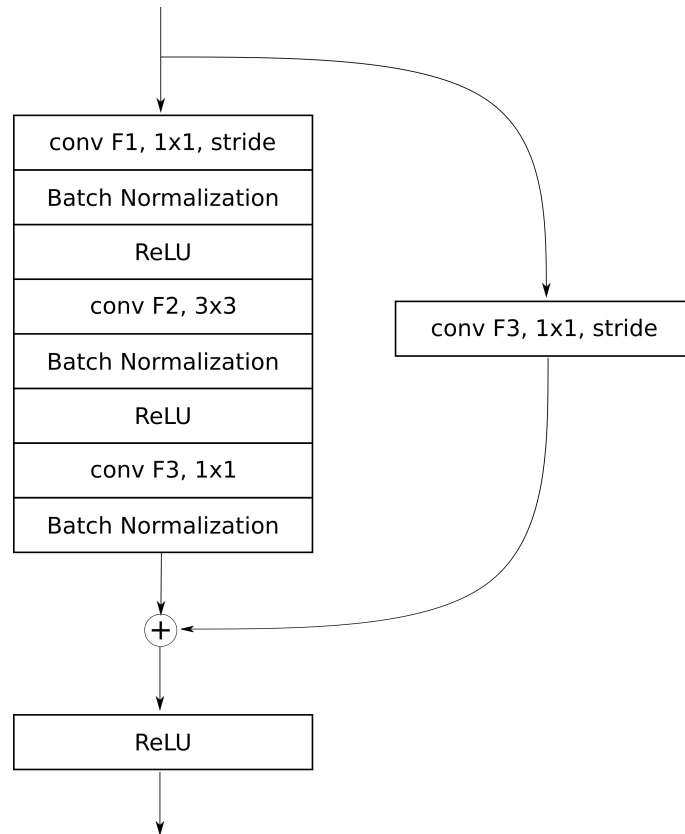


Figure 2.5: A convolutional block. One difference between convolutional blocks and identity blocks is the presence of convolutional layers in the shortcut connection. This allows the number of feature maps at the output of the block to be different from the block input because the number of feature maps can be adjusted by this layer. Another difference is the possibility of reduction in the resolution of feature maps. The first convolutional layer may apply strided convolutions that reduce feature map size. The same strides have to be applied to the convolutional layer at the shortcut connection. This is done so that the shortcut connection output dimensions match the output dimensions of last convolutional layer in the block. For the same reason, the number of filters of the convolutional layer in the shortcut connection has to match the number of filters in the third convolutional layer.

Chapter 3

Method

In this chapter a method to solve the problem of predicting sound pressure level values from video frames is presented. First, we define sound pressure level. Then, we extract the audio of an MPEG video and save it into a wave file. From the wave file audio samples, a vector of sound pressure level values is computed and saved as a binary numpy file. Each sample of this vector is generated by the calculation of sound pressure level values around a time instant t every one second. For every t when an sound pressure level value is computed, a frame is extracted from the MPEG video and saved to a binary numpy file. A pair of files is created: one vector of sound pressure level values for each t and a sequence of frames for each t . This procedure is repeated for different MPEG videos so that for each video a new vector of sound pressure level values and a new sequence of frames is also saved. This set of pairs of synchronized sound pressure level values (targets) and frames (inputs) from different videos define the dataset used for training, validation and test.

After that, 64 models are created based on the pretrained networks ResNet50 and VGG16. The FC layers of these models are removed and from that several architectures are created. These new architectures include the addition of a GAP layer and/or the addition of a hidden FC layer. All created structures share the property that their final layer is a single linear neuron. Thus, the neural networks added on top of the convolutional parts of pretrained convolutional layers are: FC, FC+FC, GAP+FC, GAP+FC+FC. For each model variation a set of hyperparameters such as type of optimizer and learning rate are tested. Models are trained on the dataset to predict sound pressure values and their performances is evaluated in terms of the MSE of the network predictions with respect to the true sound pressure level values. From a pool of models, the best model is selected based on the lowest MSE. Models with the GAP+FC structure allow the generation of a CAM that enables audio source visualization. However, subtle changes in CAM, that are related to the audio source location in the image, are faint if compared to the mean CAM. So an online exponential moving average CAM is computed to approximate a time-

varying mean CAM. Then, this exponentially-weighted average is subtracted from the CAM. The result from the subtraction is multiplied by a gain to form a matrix that is upsampled and then displayed in the green channel of the original image. This is done because the red and blue channels are sufficient to show the original scene information, so the new image with its green channel swapped displays the object localization and the original scene together. These amplified variations allow visualization of objects that the network associates with large sound pressure level values.

In summary, a dataset that associates individual video frames with scalar sound pressure levels is created. Based on this dataset, 64 models are trained and two models are chosen: one for inference and one for CAM visualization. The inference model is chosen as the one with the lowest validation MSE value. The CAM-generating model is chosen among the models with the GAP+FC structure and the lowest validation MSE value. The generated CAM is filtered for visualization to evaluate the hypothesis that trained models associate image objects with sound pressure levels. Finally, the model selected for inference is cross-validated using six train/validation folds from five videos and tested on five other videos. The sound pressure values are then converted to L_{eq} with $t_2 - t_1$ set to one minute, and a correlation between the true one minute L_{eq} and the predicted one minute L_{eq} is computed.

3.1 Dataset Generation

Ten videos were created from a camera with a mono microphone pointing towards a busy crossroad. Figure 3.1 illustrates the recording scheme: a camera films audio and video 50 meters away from the recorded area. Because the microphone is a significant distance (50 meters) apart from the region of interest, the final recorded audio is vulnerable to noise from sound sources next to the microphone. The ten videos are named according to their original filenames: M2U00001.MPG, M2U00002.MPG, M2U00003.MPG, M2U00004.MPG, M2U00005.MPG, M2U00006.MPG, M2U00007.MPG, M2U00008.MPG, M2U00012.MPG, M2U00014.MPG. In the result section, the name of the videos in the cross-validation folds are defined according to the final number in these filenames. These RGB videos were downsampled from the original 720×480 resolution to 240×240 and their audio information were extracted into wave files with ffmpeg [25]. The downsampling reduces computational cost required for future neural network processing and the storage space required to save the dataset. In addition, the Keras implementation of the ResNet50 model requires inputs with resolution larger than 180×180 , so the horizontal downsampling factor of six and the vertical downsampling factor of four

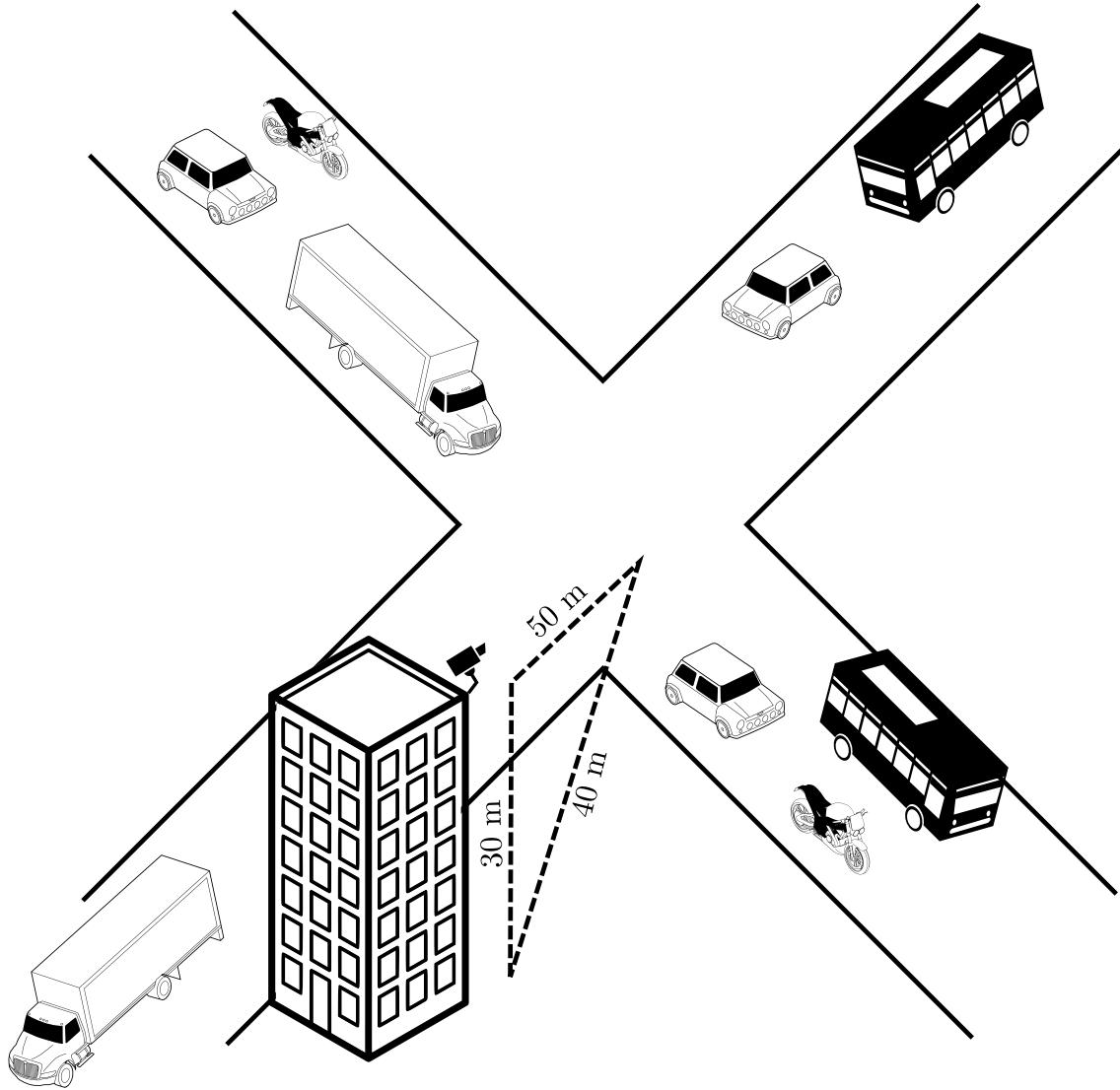


Figure 3.1: Representation of the recording scheme. The camera is positioned 50 meters from the recorded area positioned in a building 40 meters away from the crossroad at 30 meters high.

are chosen for the final resolution of 240×240 . To downsample, for each one of the ten videos the following command is run from inside a Python [26] script:

```
$ ffmpeg -i input.MPG -s 240x240 -c:a copy output.MPG
```

where input.MPG is the input 720×480 video and output.MPG is the final down-sampled 240×240 video. The argument `-s 240x240` sets the output resolution, `-c:a` copy defines that the audio from the input file will be copied to the output file without filtering. Samples of the ten videos are illustrated in Figure 3.3. To extract the audio the following script is run:

```
$ ffmpeg -i input.MPG -f wav -ar 48000 -ab bit_rate -vn output.wav
```

where input.MPG is the original video and output.wav wave file, `-ar 48000` sets the audio sample rate to 48000, which is the original value, `-ab bit_rate` sets the audio bit rate to `bit_rate`, that is the original `bit_rate` value and `-vn` means to disable video recording.

From the video and audio files, at t we define a pair of related objects: the input frame $F_t \in \mathbb{R}^{240 \times 240 \times 3}$ and the sound pressure level $S_t \in \mathbb{R}$. We define S_t as:

$$S_t = \ln \left(\frac{1}{M} \sum_{k=t-t_b}^{t+t_f} I_k^2 \right), \quad (3.1)$$

where I_k is the value of an audio sample from the respective wave file at k . Parameters t_b and t_f are respectively the backward and forward time limits considered for S_t . The number of samples I_k between time instants $(t - t_b)$ and $(t + t_f)$ is M . S_t is the natural log of the average of the square of audio samples between $(t - t_b)$ and $(t + t_f)$. The camera microphone was not initially calibrated so sound pressure values (S_t) are not in dB unless stated otherwise. Posterior to the data gathering the microphone was calibrated and the calibration formula was:

$$S_p = 4.12S_t + 5.84, \quad (3.2)$$

where S_p is the calibrated sound pressure in dB.

Initially, four videos are selected for neural network training and validation: three videos for training and one video for validation. In the training set, two videos (videos 1 and 3) are from a daytime traffic scene and one video (video 6) is from a night-time traffic scene. Values for t_b and t_f are set to 300 ms. The validation set consists of frames from a single night-time video. Frames are sampled once every second, which yields 1183, 1242, 1325 and 1280 frames from each video respectively. After the first model structure is selected, it is cross-validated with folds based on videos 1, 2, 3, 5 and 6. The final test set consists of videos 4, 7, 8, 12, 14.

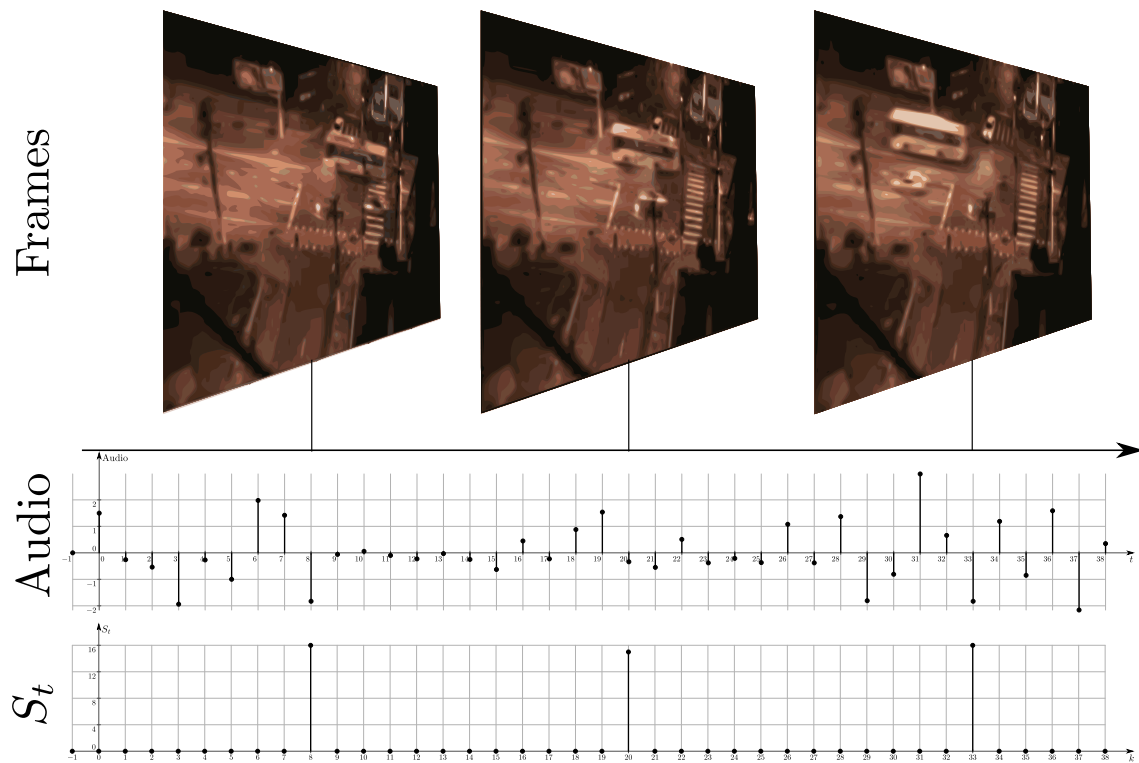


Figure 3.2: Schematic representing the dataset generation. First row represents incoming frames one at a time with time evolving to the right. The second row represents the audio stream timeline. Around t of each frame F_t , Equation 3.1 is applied to all audio samples from $t - t_b$ and $t + t_f$ and that generates the values S_t in the third row. To each frame F_t one value S_t is associated.

Daytime S_t sample sequences are clearly different from night-time S_t sample sequences: daytime S_t values range from 12 to 17, and night-time S_t values range from 10 to 16. In order to be applied to video sequences shot at different locations, the models trained on the previously described dataset might require the application of different output offset values. Without any further visual clue, sound pressure averages change from one particular environment to another one.

Challenges in this dataset include label noise from extraneous audio sources with no corresponding object in the video. For example, at times buses might break outside of the crossroad frame, which causes a peak in the S_t sample sequence without the presence of the bus itself in the image. Also, cars occasionally accelerate abruptly, and loud skid noise is heard without significant change in the video frame. Another issue arises from human voices from people near the microphone but not present in the screen itself. In two videos (videos 12 and 14) present in the final test set, a traffic warden whistle causes frequently peaking S_t values. Except for these two videos, problems are not observed to be common enough to significantly impair neural network training.

3.2 Conversion from Sound Pressure to Longer

L_{eq}

To convert the recorded sound pressure S_t to an L_{eq} with longer dt, the S_t is first converted to S_p (sound pressure in dB) with Equation (3.2). Then, Equation (3.2) is inverted as to calculate the integral as a function of L_{eq} :

$$\int_{t_1}^{t_2} \frac{p^2(t)}{p_{ref}^2} dt = 10^{\frac{L_{eq}}{10}} (t_2 - t_1). \quad (3.3)$$

The calculated values of S_p are then replaced in Equation (3.3) as the L_{eq} , with $(t_2 - t_1)$ as the value originally used to compute the S_t . In this work, we set $(t_2 - t_1)$ to 600 ms. To elongate the time interval, the integral from t_1 to t_2 is added to the integral from t_2 to t_3 , to calculate an elongated integral:

$$\int_{t_1}^{t_2} \frac{p^2(t)}{p_{ref}^2} dt + \int_{t_2}^{t_3} \frac{p^2(t)}{p_{ref}^2} dt = \int_{t_1}^{t_3} \frac{p^2(t)}{p_{ref}^2} dt. \quad (3.4)$$

To integrate over the specified L_{eq} time interval, this process is repeated. Finally the L_{eq} with a larger time span is calculated with Equation (1.1) but with a new interval $t_3 - t_1$ as the new longer time span. The conversion is used to translate the predicted S_t and true S_t to an L_{eq} from an original time interval of 600 ms to one minute.

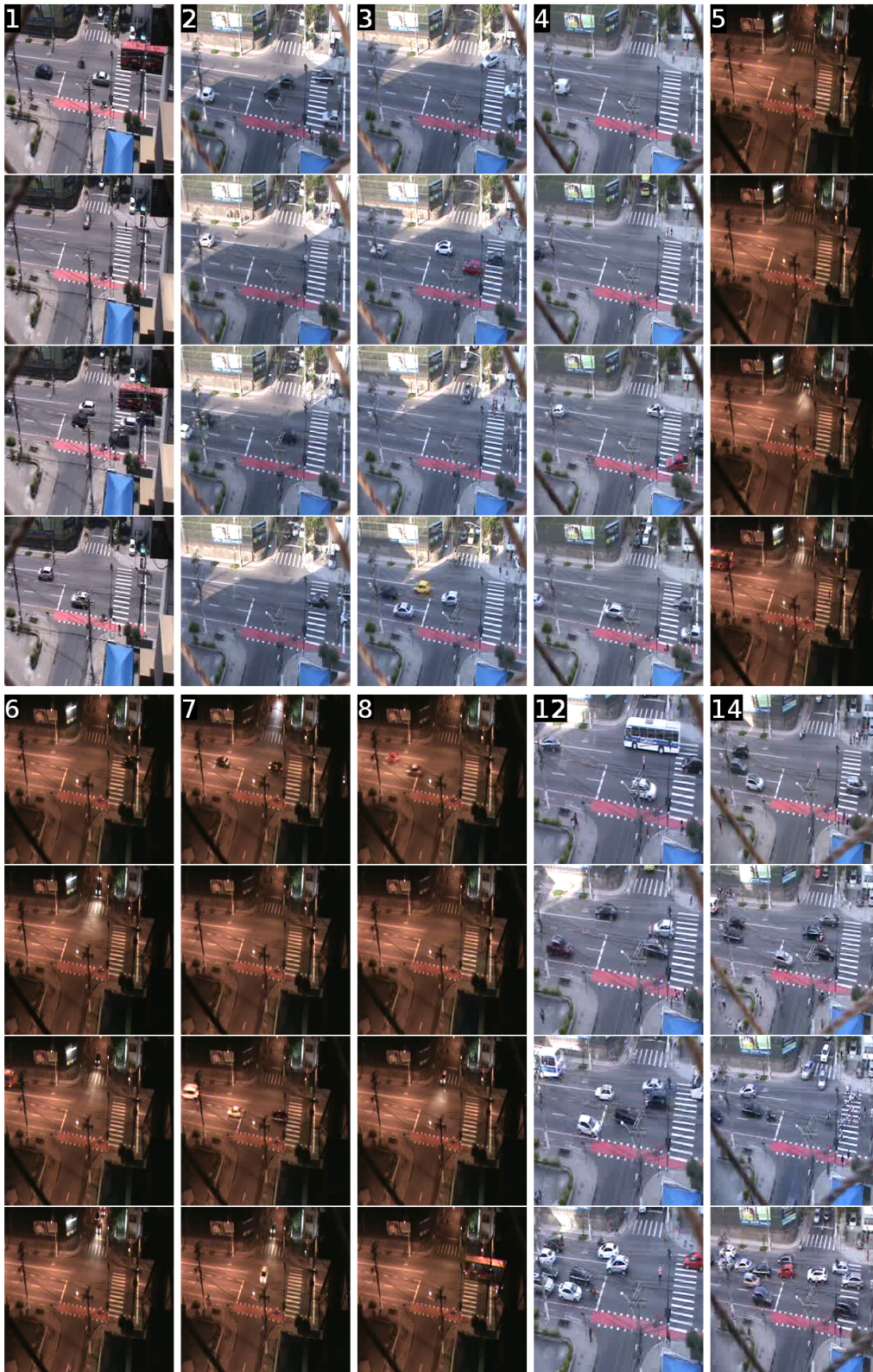


Figure 3.3: Samples from the ten videos. Every set of four rows in each column are samples from a distinct video. The number in the upper left corner is the video name as described in section 3.1.

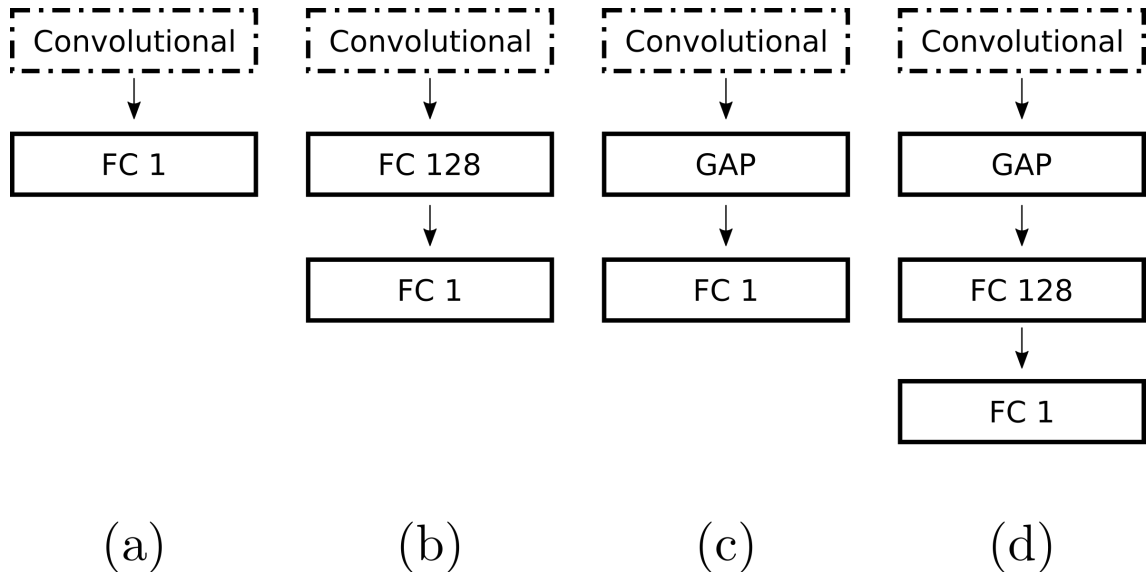


Figure 3.4: Models tested for prediction of sound pressure level values. The convolutional block represents either the convolutional part of VGG16 or ResNet50. In the case of VGG16, the output of the last convolutional layer lies in $\mathbb{R}^{7 \times 7 \times 512}$. In the first model (a), the FC layer with a single neuron contains $7 \times 7 \times 512 + 1 = 25089$ parameters. In the second model (b), the first FC layer contains $7 \times 7 \times 512 \times 128 + 128 = 3211392$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the third model (c), the FC layer contains $512 + 1 = 513$ parameters. In the fourth model (d), the first FC layer contains $512 \times 128 + 128 = 65664$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the case of ResNet50, the number of feature maps in the last convolutional layer quadruples, so that the last convolutional layer output size is $7 \times 7 \times 2048$. In the first model (a), the FC layer with a single neuron contains $7 \times 7 \times 2048 + 1 = 100353$ parameters. In the second model (b), the first FC layer contains $7 \times 7 \times 2048 \times 128 + 128 = 12845184$ parameters and the second FC layer contains $128 + 1 = 129$ parameters. In the third model (c), the FC layer contains $2048 + 1 = 2049$ parameters. In the fourth model (d), the first FC layer contains $2048 \times 128 + 128 = 262272$ parameters and the second FC layer contains $128 + 1 = 129$ parameters.

3.3 Models

Using the generated dataset we train 64 models. They were based on the convolutional parts of ResNet50 and VGG16. Four combinations of new FC layers are tested: a single FC, FC+FC, GAP+FC and GAP+FC+FC (Figure 3.4). For each combination, the convolutional part of the model comes from either VGG16 or ResNet50 for a total of 8 models. The hidden 128 FC layer is followed by a hyperbolic tangent activation function.

3.4 Training

To train the neural networks the input frames are first pre-processed, then model parameters are selected by minimizing the cost function with an optimizer. The VGG16 and ResNet50 models had been already pretrained in the ImageNet dataset with a pre-processing that subtracts channels R, G and B by 103.939, 116.779, 123.68 respectively. Instead, we follow the pre-processing steps that are described in [18]. Those pre-processing steps first compute the mean and the standard deviation for each image channel taking the entire training data into account. Then each image channel is subtracted by its respective mean and divided by its respective standard deviation:

$$\begin{aligned} R' &= \frac{R - \mu_R}{\sigma_R} \\ G' &= \frac{G - \mu_G}{\sigma_G} \\ B' &= \frac{B - \mu_B}{\sigma_B} \end{aligned} \tag{3.5}$$

where μ_R, μ_G, μ_B are the channel-wise RGB means over the training data and $\sigma_R, \sigma_G, \sigma_B$ are the channel-wise RGB standard deviations over the training data. The pre-processed frames are fed into the model. The network output is the predicted sound pressure level values for each frame F_t at t . For each pair of (F_t, S_t) , predictions $f(F_t, \theta)$ are compared with the true sound pressure level value S_t and then averaged:

$$J(\theta) = \sum_{t=1}^N \frac{\|f(F_t, \theta) - S_t\|_2^2}{N} \tag{3.6}$$

Parameters θ are selected to minimize the cost function $J(\theta)$ corresponding to the MSE loss. For that, two optimizers are tested: SGD with momentum and Adam. Also, for each optimizer, four initial learning rates are tested: 0.1, 0.01, 0.001, 0.0001. In all these cases the learning rates are reduced twice, during training, by a factor of 0.3: first at epoch 30 and then again at epoch 50 (Table 3.1). All models are trained up until epoch 70. The mini-batch size is 32, which yields approximately 115 optimization updates per epoch or approximately 7700 optimization updates for the whole training. For each training setting (optimizer, initial learning rate and model), the model with the lowest validation loss among all epochs is saved. During inference, no online audio data is made available to the model at each time instant t , so the network has to predict sound pressure levels with only the current frame and no temporal information.

The total number of models trained for each experiment (comparison between settings) is 64. One difficulty encountered during the project was on how to organize

Table 3.1: Four learning rate schedules by epoch. Each column is one schedule.

Epoch	Learning Rates			
epoch <30	0.1	0.01	0.001	0.0001
30 ≤ epoch <50	0.03	0.003	0.0003	0.00003
epoch ≥ 50	0.009	0.0009	0.00009	0.000009

this large number results so that performance obtained from different settings could be quickly and objectively compared. To solve that problem, a L^AT_EX report is automatically generated from the experiment results, including data such as: validation loss, the name of video files included in the dataset, the frame sampling rate, the model structure, the saved model filename, the training/validation loss curves, and an image of network predictions vs. true sound pressure level values for training and validation. This turned feasible an objective comparison of the performance of the 64 models.

3.4.1 Sound Source Detection

Since the network has no access to online audio data, it is expected that the model has to detect, within the current image, objects that are relevant for sound pressure value prediction. To test this hypothesis, the CAM technique is applied. Models with the GAP+FC structure can be used to generate CAMs that identify image locations that are associated with large network outputs [19]. So if the network truly maps some objects visible in the image to high network outputs, they should be highlighted in the CAM.

The network used for source detection is based on the convolutional part of VGG16. This network is chosen because of its lowest validation MSE of 1.196. This model follows the structure of the third network (from left to right) in Figure 3.4: a convolutional model (from VGG16) and a GAP+FC structure. This model is one of the 64 networks trained to predict S_t values by minimizing MSE between the network outputs and the true S_t values.

The last convolutional layer of VGG16 yields 512 7×7 feature maps, which constrains object detection resolution to a 7×7 grid. To improve CAM visualization, we upsample the 7×7 grid to 240×240 by nearest neighbor interpolation, and then replace the green channel of the input frame by the CAM. We observed that some portions of the CAM remain fixed for several consecutive frames. These fixed values have a large magnitude, in comparison to CAM variations that highlight the detected object, which makes the green channel variations too subtle to be visualized. So we first subtract that average CAM (AC) offset, and then apply a gain α to the green channel, in order to magnify the CAM variations. We compute an online 7×7 AC

by:

$$AC[n] = AC[n - 1]\lambda + C[n - 1](1 - \lambda) \quad (3.7)$$

$$DC[n] = \alpha(C[n] - AC[n]), \quad (3.8)$$

where $C[n] \in \mathbb{R}^{7 \times 7}$ is the CAM at frame n , $AC[n] \in \mathbb{R}^{7 \times 7}$ is the AC at frame n , $DC[n] \in \mathbb{R}^{7 \times 7}$ is the displayed CAM at frame n , $\lambda \in (0, 1)$ and $\alpha \in \mathbb{R}_+$. The closer λ is to 1, the longer the effective sample sequence that is taken into account for CAM averaging. The higher α is, the higher the magnification of the highlighted region but also the larger that unwanted noise spots become. An exceedingly high value of α could also saturate the green channel, and that could hide DC variations, that would otherwise differentiate objects associated with high sound pressure from objects associated with low sound pressure. We choose $\lambda = 0.95$ and $\alpha = 10$. For sound source detection we do use temporal information to filter the CAM, however this method only requires frames and no online audio data. A downside of this method is that if a sound-generating source does not move within several frames (which is typical for noisy vehicles stopped at traffic signal lights), then that source might be filtered out as part of the AC . Larger values for λ might mitigate this type of issue.

3.4.2 Model Validation

After the models are optimized according to the training setup, the best of the 64 models is cross-validated. This is done by first separating a test set from the ten available videos. Five of the ten videos are considered part of the test set: three daytime videos (videos 4, 12, and 14 in the numbering system of Figure 3.3) and two night-time videos (videos 7 and 8). In the remaining five videos, three are daytime (videos 1, 2 and 3) and two are night-time (videos 5 and 6). The train/validation set is arranged into six folds for cross-validation. Each fold is composed of two sets: the training set and the validation set. The folds are selected so that the validation set in every fold contains one night-time video and one daytime video. So if the three daytime videos are D_1 , D_2 and D_3 and the night-time videos are N_1 and N_2 , then the folds at each iteration are as described in Table 3.2. D_1 is video 1, D_2 is video 2, D_3 is video 3, N_1 is video 5 and N_2 is video 6. The initial learning setup is equal to that of the best of the 64 models. The learning rate is reduced by a factor of ten every time the validation loss does not decrease for 15 consecutive epochs. Then, for each fold, the model with lowest MSE has the correlation of its output with the true sound pressure level computed. Finally, one model from the six folds is chosen according to its MSE and correlation, and it is evaluated in the test set.

Table 3.2: Table of the dataset folds. D_i is a daytime video with index i . N_i is a night-time video with index i . Each row is one fold.

Fold	Training set			Validation set	
1	D_1	D_2	N_1	D_3	N_2
2	D_1	D_3	N_1	D_2	N_2
3	D_2	D_3	N_1	D_1	N_2
4	D_1	D_2	N_2	D_3	N_1
5	D_1	D_3	N_2	D_2	N_1
6	D_2	D_3	N_2	D_1	N_1

Chapter 4

Results and Discussion

Results for the prediction of sound pressure level values using image frames as neural network inputs are presented. These results correspond to predictions in the validation video 8 by two models: the first with the lowest prediction MSE over all models and the second with the lowest prediction MSE among models with the GAP+FC structure. The *DC* is presented for the GAP+FC model to illustrate regions in the image associated with large network output values. This model (with the GAP+FC structure) allows the application of CAM and testing whether the network associates specific objects with large sound pressure level values by observing the highlighted regions. Table 4.1 shows all the models trained with their respective training setup ordered by the lowest validation loss first. At the lower end of the table, models with values > 1000 in *t_loss* or in *v_loss* diverged. In Table 4.1, *Network* specifies the base convolutional model. *Opt* defines the optimization algorithm: either Adam or SGD. The *Gap* column displays if the full network (convolutional part and FC layers) contains a GAP after the last convolutional layer. *FC* stands for how many neurons are there in the hidden fully connected layers: if the value displayed in this column is “None”, then there is no hidden FC layer. *lr_list* is the list of scheduled learning rates as listed in Table 3.1. *v_loss* is the validation loss of the model with the lowest mean squared error among all epochs (for a single set of settings). *t_loss* is the training loss associated with that model. Models whose columns *GAP* and *FC* contain respectively the labels “True” and “None” are the ones where CAM visualizations are applicable. So, as observed in Table 4.1, the network with the lowest MSE has index 35. Model 35 is composed by the convolutional part of VGG16, is optimized with Adam, has a GAP layer and a hidden layer with 128 neurons. Its schedule has an initial learning rate of 0.001 with training loss 0.461 and validation loss 1.138. In comparison, the first model where the CAM visualization is applicable has index 27, uses the same optimizer, an initial learning rate of 0.01 and validation loss 1.196. In Section 4.1, the training/validation loss curves of model 35 are presented together with the predictions of sound pressure level values. Section

4.2 shows training/validation loss curves of model 27 and its predictions of sound pressure level values. Section 4.2.1 presents the *DC*, generated with model 27, that highlights image regions associated with higher prediction values.

Generally, models based on the convolutional part of VGG16 achieved lower training loss and lower validation loss. Most models built with ResNet50 either achieved a high validation loss or diverged. The absence of GAP after the last convolutional layer is associated with lower training losses in several models (based on VGG16 or ResNet50). For example, model 5 attained a training loss of 0.259: roughly half of the training loss of model 35. However, its validation loss is still similar to that of model 1: a model with very similar settings but lower parameter count. The pattern of lower training losses without the counterpart lower validation losses in models without GAP repeats through models 55, 29, 63 and other models. This suggests that the presence of GAP does not negatively impact model performance (measured with MSE) in spite of significantly reducing parameter count. The Adam optimizer is slightly more common among the top ten models (six out of ten). Among these models, two have initial learning rate 10^{-4} , one 10^{-3} , two 10^{-2} and one 10^{-1} . In comparison, three of the four models optimized with SGD (in the overall top ten) have an initial learning rate of 10^{-1} and one with initial learning rate 10^{-2} . The SGD optimizer seems to require more specific tuning to attain lower validation losses.

Table 4.1: Each row corresponds to one training setup. Column Network has the name of the base convolutional network. Column Opt has the name of the optimizer. Column GAP defines the presence or absence of GAP after the last convolutional layer. Column FC is the number of neurons in the hidden layer. The value “None” in this column specifies the absence of a hidden layer. lr_list is the schedule of learning rates with reductions in epochs 30 and 50. t_loss and v_loss are respectively the training loss and validation loss.

	Network	Opt	GAP	FC	lr_list	t_loss	v_loss
35	VGG16	Adam	True	128	[0.001, 0.0003, 9e-05]	0.461	1.138
19	VGG16	Adam	True	128	[0.01, 0.003, 0.0009]	0.450	1.140
1	VGG16	SGD	True	128	[0.1, 0.03, 0.009]	0.524	1.167
5	VGG16	SGD	False	128	[0.1, 0.03, 0.009]	0.259	1.169
27	VGG16	Adam	True	None	[0.01, 0.003, 0.0009]	0.576	1.196
11	VGG16	Adam	True	None	[0.1, 0.03, 0.009]	0.517	1.209
51	VGG16	Adam	True	128	[0.0001, 3e-05, 9e-06]	0.562	1.211

Continued on next page

Table 4.1: Training and validation loss for each configuration of the trained models.

	Network	Opt	GAP	FC	lr_list	t_loss	v_loss
55	VGG16	Adam	False	128	[0.0001, 3e-05, 9e-06]	0.390	1.214
29	VGG16	SGD	False	None	[0.01, 0.003, 0.0009]	0.420	1.223
9	VGG16	SGD	True	None	[0.1, 0.03, 0.009]	0.568	1.231
63	VGG16	Adam	False	None	[0.0001, 3e-05, 9e-06]	0.257	1.233
39	VGG16	Adam	False	128	[0.001, 0.0003, 9e-05]	0.461	1.243
21	VGG16	SGD	False	128	[0.01, 0.003, 0.0009]	0.355	1.252
43	VGG16	Adam	True	None	[0.001, 0.0003, 9e-05]	0.613	1.257
13	VGG16	SGD	False	None	[0.1, 0.03, 0.009]	0.355	1.262
47	VGG16	Adam	False	None	[0.001, 0.0003, 9e-05]	0.666	1.284
17	VGG16	SGD	True	128	[0.01, 0.003, 0.0009]	0.613	1.300
37	VGG16	SGD	False	128	[0.001, 0.0003, 9e-05]	0.553	1.301
45	VGG16	SGD	False	None	[0.001, 0.0003, 9e-05]	0.515	1.306
25	VGG16	SGD	True	None	[0.01, 0.003, 0.0009]	0.663	1.409
31	VGG16	Adam	False	None	[0.01, 0.003, 0.0009]	0.453	1.548
59	VGG16	Adam	True	None	[0.0001, 3e-05, 9e-06]	0.825	1.699
10	ResNet50	Adam	True	None	[0.1, 0.03, 0.009]	16.002	1.720
33	VGG16	SGD	True	128	[0.001, 0.0003, 9e-05]	0.816	1.753
41	VGG16	SGD	True	None	[0.001, 0.0003, 9e-05]	0.821	1.790
2	ResNet50	Adam	True	128	[0.1, 0.03, 0.009]	8.873	1.792
3	VGG16	Adam	True	128	[0.1, 0.03, 0.009]	3.067	1.792
6	ResNet50	Adam	False	128	[0.1, 0.03, 0.009]	1.977	1.793
7	VGG16	Adam	False	128	[0.1, 0.03, 0.009]	1.537	1.797
34	ResNet50	Adam	True	128	[0.001, 0.0003, 9e-05]	0.528	1.808
61	VGG16	SGD	False	None	[0.0001, 3e-05, 9e-06]	0.850	1.824
4	ResNet50	SGD	False	128	[0.1, 0.03, 0.009]	0.674	1.826
53	VGG16	SGD	False	128	[0.0001, 3e-05, 9e-06]	2.095	1.839
22	ResNet50	Adam	False	128	[0.01, 0.003, 0.0009]	2.751	1.841
23	VGG16	Adam	False	128	[0.01, 0.003, 0.0009]	1.246	1.934
18	ResNet50	Adam	True	128	[0.01, 0.003, 0.0009]	1.221	2.033
38	ResNet50	Adam	False	128	[0.001, 0.0003, 9e-05]	1.155	2.322
54	ResNet50	Adam	False	128	[0.0001, 3e-05, 9e-06]	1.143	2.657
15	VGG16	Adam	False	None	[0.1, 0.03, 0.009]	19.392	5.470
26	ResNet50	Adam	True	None	[0.01, 0.003, 0.0009]	1.192	8.783

Continued on next page

Table 4.1: Training and validation loss for each configuration of the trained models.

	Network	Opt	GAP	FC	lr_list	t_loss	v_loss
0	ResNet50	SGD	True	128	[0.1, 0.03, 0.009]	0.593	10.027
50	ResNet50	Adam	True	128	[0.0001, 3e-05, 9e-06]	0.364	12.149
46	ResNet50	Adam	False	None	[0.001, 0.0003, 9e-05]	75.788	13.867
8	ResNet50	SGD	True	None	[0.1, 0.03, 0.009]	0.424	15.278
42	ResNet50	Adam	True	None	[0.001, 0.0003, 9e-05]	0.496	17.901
20	ResNet50	SGD	False	128	[0.01, 0.003, 0.0009]	0.575	27.870
32	ResNet50	SGD	True	128	[0.001, 0.0003, 9e-05]	0.658	29.182
58	ResNet50	Adam	True	None	[0.0001, 3e-05, 9e-06]	0.779	30.011
24	ResNet50	SGD	True	None	[0.01, 0.003, 0.0009]	0.616	31.649
16	ResNet50	SGD	True	128	[0.01, 0.003, 0.0009]	0.536	34.324
12	ResNet50	SGD	False	None	[0.1, 0.03, 0.009]	0.830	44.495
36	ResNet50	SGD	False	128	[0.001, 0.0003, 9e-05]	0.517	45.463
57	VGG16	SGD	True	None	[0.0001, 3e-05, 9e-06]	64.872	47.675
49	VGG16	SGD	True	128	[0.0001, 3e-05, 9e-06]	62.804	49.207
40	ResNet50	SGD	True	None	[0.001, 0.0003, 9e-05]	1.174	52.486
62	ResNet50	Adam	False	None	[0.0001, 3e-05, 9e-06]	1.110	59.933
30	ResNet50	Adam	False	None	[0.01, 0.003, 0.0009]	>1000	69.145
52	ResNet50	SGD	False	128	[0.0001, 3e-05, 9e-06]	0.842	79.702
60	ResNet50	SGD	False	None	[0.0001, 3e-05, 9e-06]	3.872	79.801
48	ResNet50	SGD	True	128	[0.0001, 3e-05, 9e-06]	22.422	81.286
44	ResNet50	SGD	False	None	[0.001, 0.0003, 9e-05]	0.525	86.197
28	ResNet50	SGD	False	None	[0.01, 0.003, 0.0009]	0.257	101.153
56	ResNet50	SGD	True	None	[0.0001, 3e-05, 9e-06]	41.233	122.410
14	ResNet50	Adam	False	None	[0.1, 0.03, 0.009]	>1000	>1000

These models were trained on a computer with an i7-6850K CPU @ 3.60 GHz, 64 GB RAM and 4 GPUS 1080Ti, one/two of which were used for training. Each model required roughly 35 minutes to train. As the convolutional part of the models were not trained, the features (outputs of the last convolutional layer) could have been cached to speed up training and reduce memory requirements. However, features were not cached at this step of the development to simplify programming efforts (this was possible because sufficient computational resources were available).

4.1 Model with the Lowest MSE

Model 35 attained the lowest validation MSE of 1.138 among all models. Figure 4.1 shows the log loss curves for model 35. The training loss rapidly reduces in the first ten epochs and then slowly reduces in future epochs. The validation loss decreases in the first epochs and mostly stalls until the final epoch.

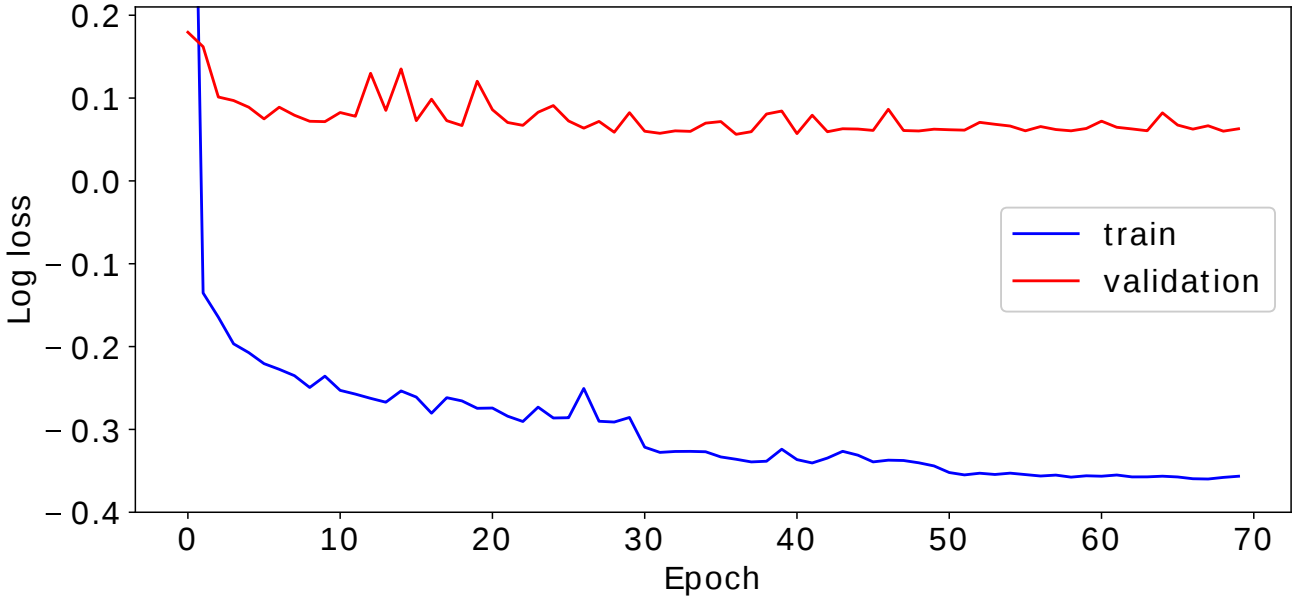


Figure 4.1: Training and validation log loss curves for model 35. The log used is base 10.

Figure 4.2 shows predictions for training and validation data. The daytime sound pressure level values are higher than the sound pressure level values during night-time. From frame 0 to frame 1180 of a daytime video, the base S_t level is around 14. From frames 1180 to 2400 of a night-time video, the base S_t level is roughly 12. From frames 2400 to 3700 of a daytime video, the base S_t level is again around 14. The model trained with input frames from daytime videos and night-time videos correctly predicts base S_t values of a night-time validation video. This implies that the model correctly identifies frames from night-time videos.

Figure 4.3 illustrates the prediction of model 35 in the validation data. The network predicts average sound pressure values with correlation 0.607. It is capable of following trending S_t samples. The network had worse performance during low signal intensity intervals. It is possible that in spite of the significant differences between the real S_t and the network prediction during low sound pressure intervals (Figure 4.3, frames 3880 to 3940, among other intervals) the image had no recognizable sound-generating objects throughout these intervals. Figure 4.3 shows that

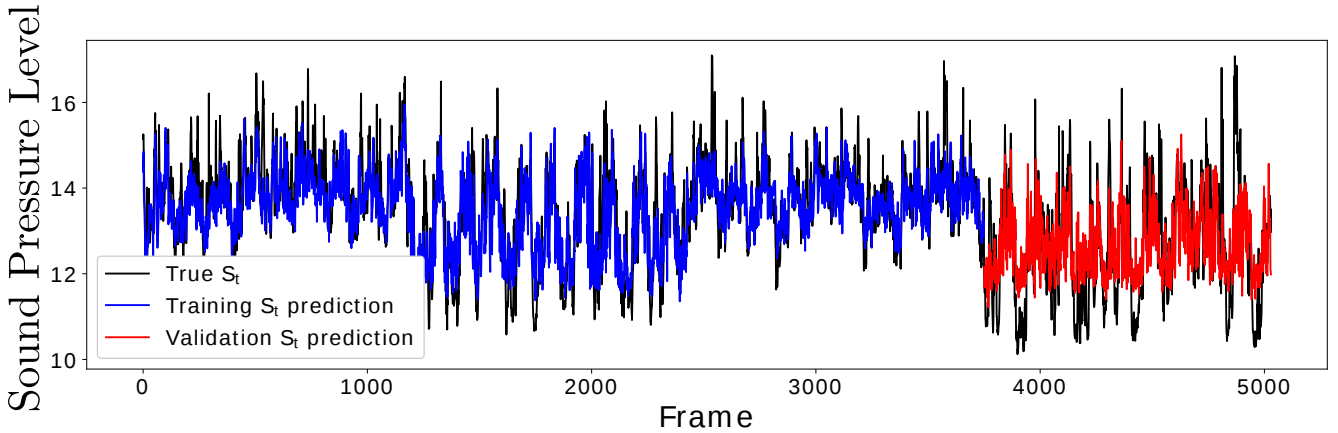


Figure 4.2: Predictions of sound pressure level S_t for model 35 in training and validation data. Sound pressure levels S_t can be converted to sound pressure levels in dB through Equation 3.2.

in spite of label noise, the network learned correct maps between visual objects and salient S_t values. The S_t values at frames 4800 and 4860 are largely missed by the network. The corresponding video shows an event in which a bus breaks, which happens outside of the camera view, creates the spike at 4800, then a motorbike skids at 4860, and that causes the second spike at that time. As expected, it is not possible to correctly infer S_t values in those cases, because the associated objects are not present in the image.

4.2 Model GAP+FC with Lowest MSE

Model 27 has the lowest MSE among models with the GAP+FC structure. The network attained an MSE of 1.196 and predicts sound pressure level values with correlation 0.592. Figure 4.4 shows the training loss and validation loss over the 70 training epochs. The training loss rapidly decreases over the first 20 epochs and then stalls until the first learning rate reduction at epoch 30. The validation loss reduces after the first ten epochs then stops improving.

Figure 4.5 shows the predictions in training data and in validation data. This model is also capable of discerning between daytime and night-time sound pressure level values as it is seen from it correctly predicting the base S_t level in the validation video 8. The detail of S_t predictions in Figure 4.5 shows a very similar overall result to that of model 35. Trending levels of sound pressure level values are followed by the model. However, the model fails to predict peaks at 4800 and 4860, similarly to the result of model 35. Model 27 cannot correctly evaluate the S_t at these positions either, since the corresponding audio-generating sources are not visible in the image

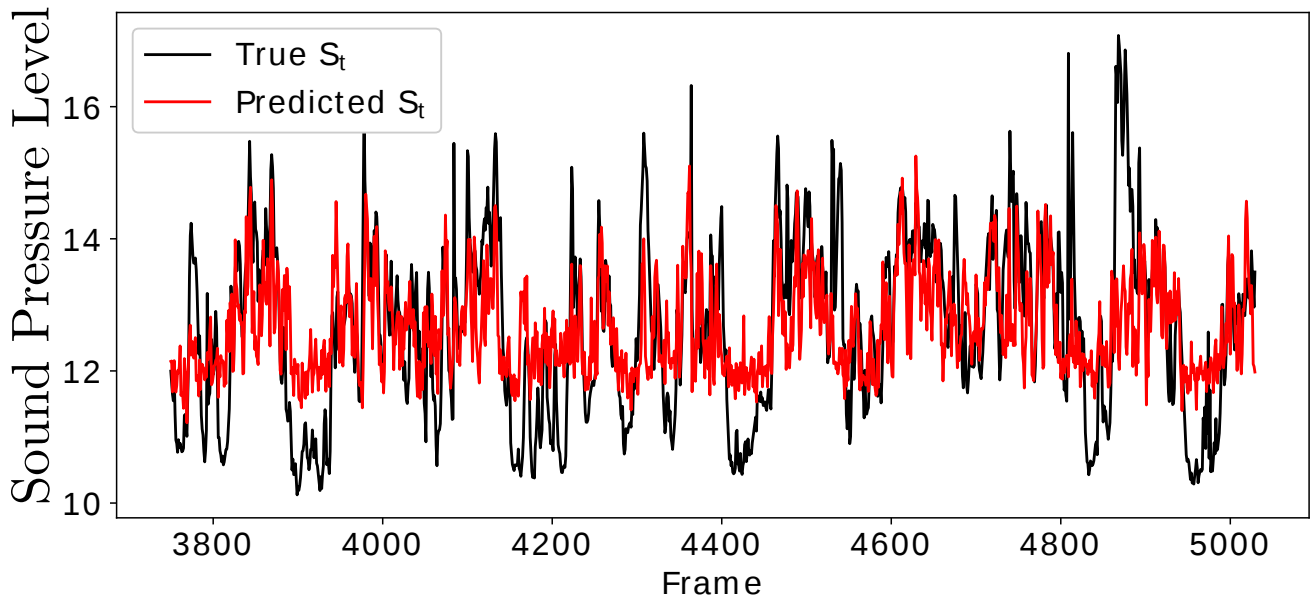


Figure 4.3: Detail of the predictions in the validation data of model 35.

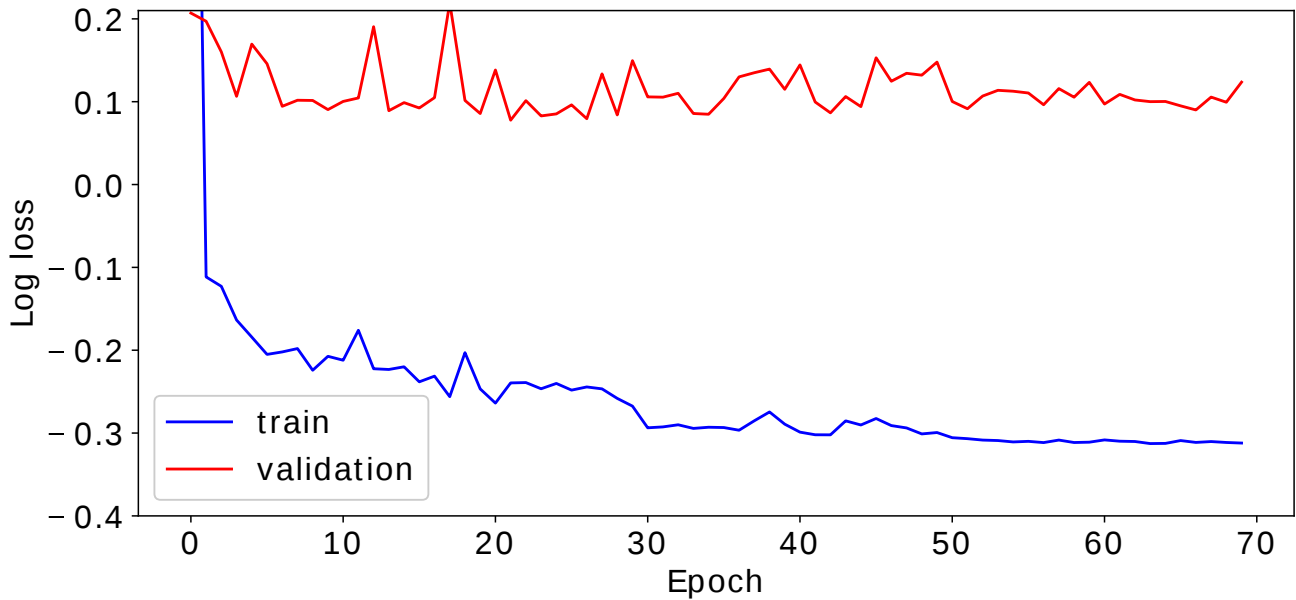


Figure 4.4: Training and validation log loss curves for model 27. The log used is base 10.

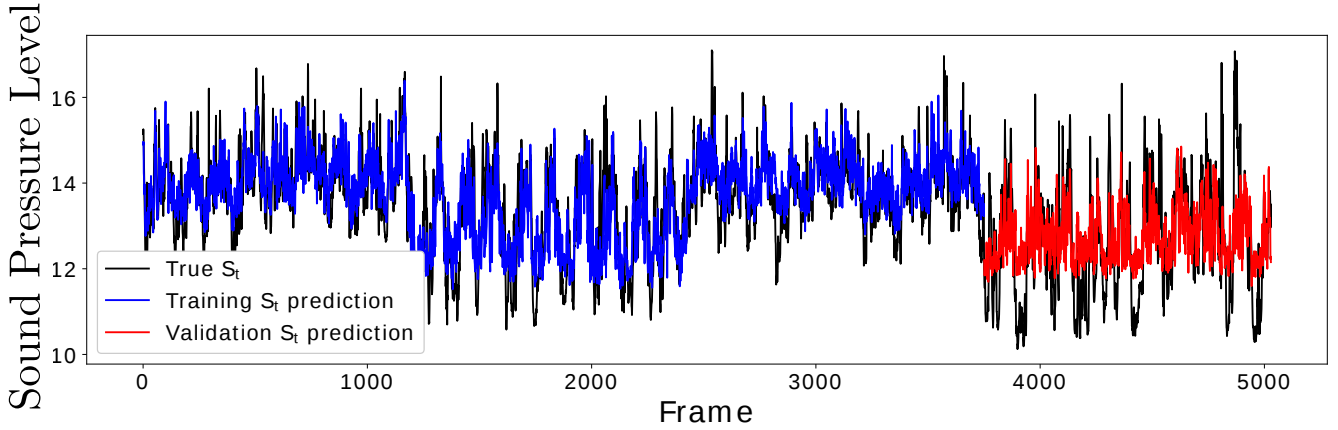


Figure 4.5: Predictions of sound pressure level S_t for model 27 in training and validation data.

itself.

4.2.1 Sound Source Detection

Model 27 contains the structure GAP+FC and that allows the application of CAMs to visualize the regions in the image responsible for larger output values. This allows the verification of the hypothesis that the network associates objects with S_t values to correctly predict sound pressure level values. Figures 4.7 and 4.8 show the visualization of the original frame in the left, the CAM in the middle and the filtered CAM (DC) as the green channel of the image in the right. In Figure 4.7 the network clearly focuses on the bus to predict higher sound pressure level values from this frame. In comparison, the cars in Figure 4.8 are lightly highlighted by the CAM, with the addition of unwanted green spots around the image without any clear relationship to the sound-generating objects. An example of the filtering result in the DC is illustrated in Figure 4.7. Several regions in the CAM have similarly high value, shown as a light green in the CAM visualization (image in the middle), however the image in the right only highlights the CAM variations: the filtering process removes fixed unwanted regions. When a car drives through the crossroad (Figure 4.8) it is clear the network associates lower sound pressure level values to it if compared to the bus based on the CAM visualization. This supports the hypothesis that the network discriminates between objects in the image and also associates specific objects with S_t values.

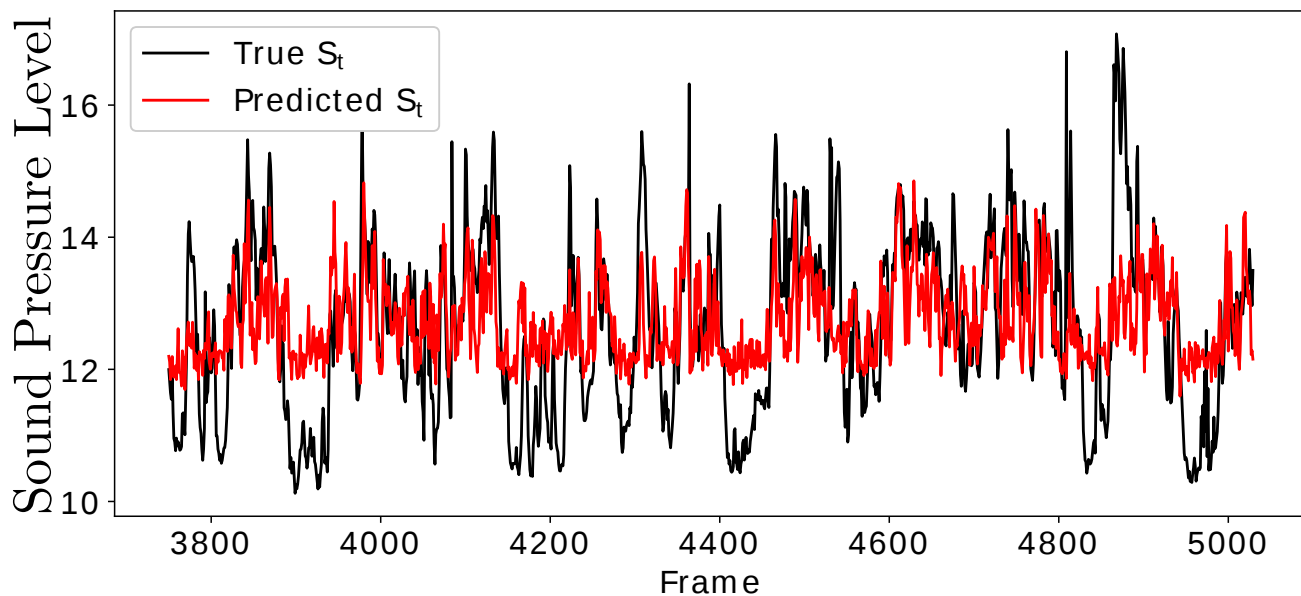


Figure 4.6: Detail of the predictions in the validation data of model 27.

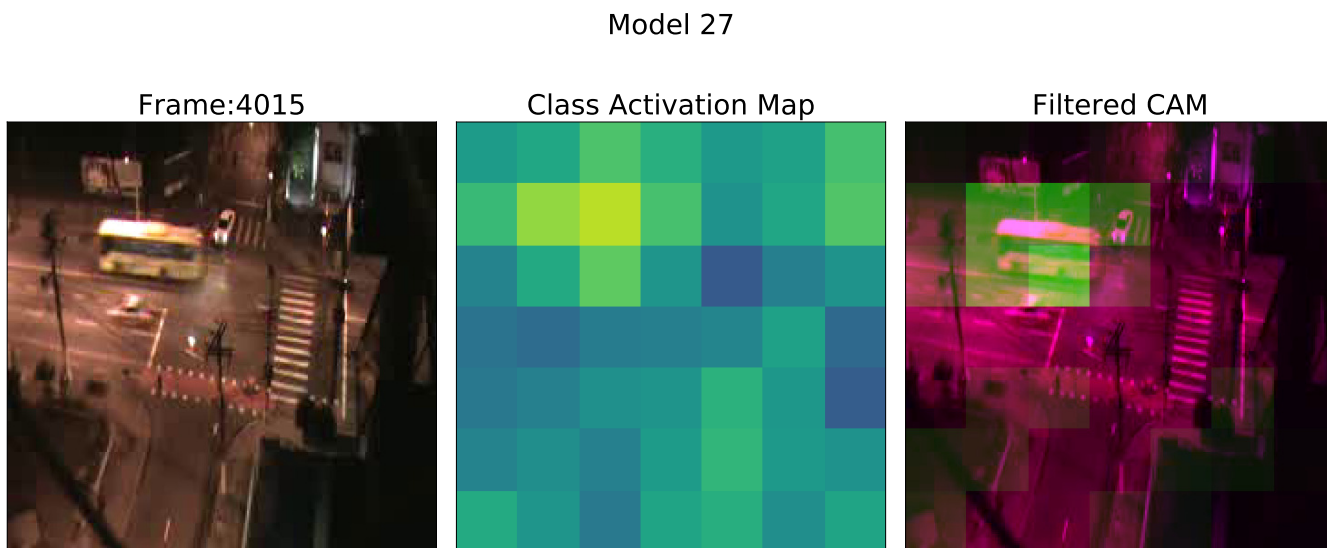


Figure 4.7: A particularly good result: when a noisy bus drives trough the crossroad.

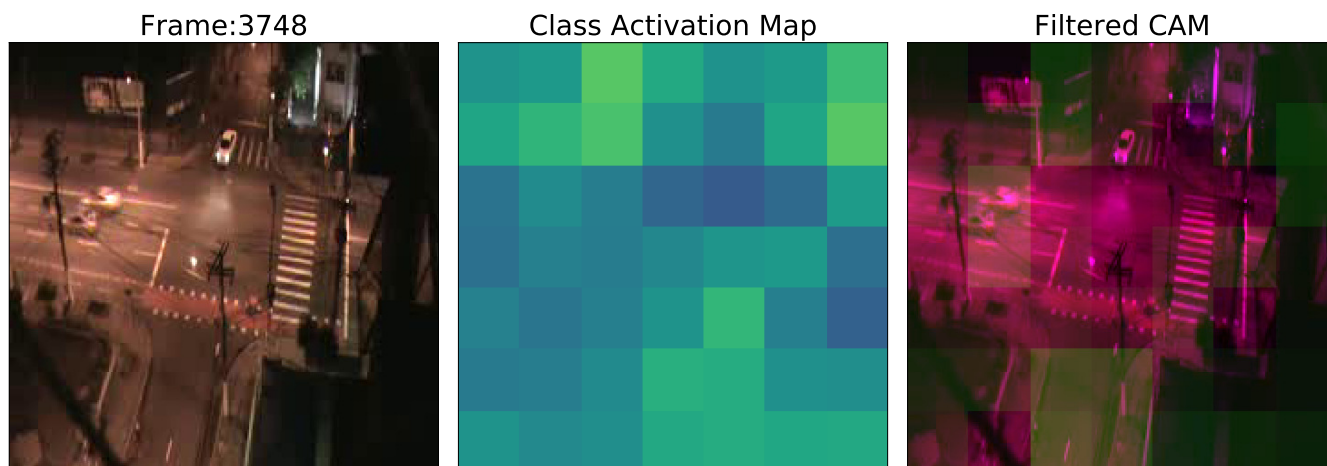


Figure 4.8: A typical example: a car drives through the crossroad with unwanted noisy spots around the image.

4.3 Model Validation

As model 35 had the lowest MSE among all models, it is chosen for cross-validation using the folds specified in Table 3.2. The correlation and MSE results for each fold are displayed at Table 4.2. The loss curves and predictions for the model within each fold are found in the appendices. It is seen from Table 4.2 that the MSE over all folds is lower than that the first training, where 64 models are present. It could be argued that since the validation video in the first training was a single night-time video, it is possible that it had less traffic. So the average number of frames without sound generating objects is expected to be smaller. Therefore, the average contribution of frames associated with lower sound pressure level values (more difficult to predict) is reduced. No single model had both the lowest MSE and the highest correlation simultaneously. The model from fold 1 is chosen as the best model from the 6 folds for its relatively high correlation (above 0.64) and low MSE. This model is then evaluated in the test set. In this stage of development, features were cached so that allowed models to train in approximately 30 seconds each in a laptop with an i7-6700HQ CPU @ 2.60 GHz, 16 GB of RAM and a GTX1060 3 GB.

Table 4.3 shows the correlations and MSE of the selected model from the fold in the test set. Predictions in videos 4, 7 and 8 attained a correlation of roughly 0.6. Predictions in daytime videos are associated with lower MSE and predictions in night-time videos are associated with higher MSE. Correlation in videos 12 and 14 are exceptionally low compared to overall results. Further investigation showed that

Table 4.2: Correlations and MSE for the validation in each fold. D_1, D_2, D_3 are the daytime videos 1, 2 and 3 respectively. N_1 and N_2 are night-time videos 5 and 6 respectively.

Fold	Training set			Validation set		Correlation	MSE
1	D_1	D_2	N_1	D_3	N_2	0.649	0.721
2	D_1	D_3	N_1	D_2	N_2	0.639	0.780
3	D_2	D_3	N_1	D_1	N_2	0.651	0.796
4	D_1	D_2	N_2	D_3	N_1	0.608	0.668
5	D_1	D_3	N_2	D_2	N_1	0.615	0.694
6	D_2	D_3	N_2	D_1	N_1	0.632	0.717

in these two videos a traffic warden constantly whistles loudly. Since that was not present in the training set, the model cannot correctly predict the sound pressure level values in this situation. This caused the correlation to be extremely lower in these two videos if compared to the other three. To test whether the model could learn to predict sound pressure level values if samples from videos with the traffic warden were present in the training set, a new separate fold is created. This new fold consists of videos 2, 5 and 14 in the training set and videos 6 and 12 in the validation set. This new fold contains samples from one video with a traffic warden in the training set (video 14) and samples from one video with a traffic warden in the validation set (video 12). In spite of that, performance did not improve. The correlation of the model predictions with the true sound pressure level values is 0.222 for video 12 and 0.629 for video 6. It is possible that the low input frame resolution made it impossible to distinguish whether the traffic warden was about to move the whistle to his mouth or not. Even then, there are moments in the video when the traffic warden is in his back, so no information about sound generation is available in the image. Other situations include moments when the traffic warden has the whistle in his mouth but only produces sound sporadically, so no visual cues could aid in sound pressure level prediction. These events may explain the low correlation of the model output with samples from video 14 (in the test set) and with video 12 (in the test set and in the new fold).

The results are further validated with the L_{eq} values with one minute time dependency (in dB and with $(t_2 - t_1) = 1$ minute converted from S_t values (not in dB, computed with a 600 ms dependency) using Equation (1.1)). L_{eq} trends are visibly followed in Figure 4.10. In Figure 4.11, the predicted L_{eq} and the true L_{eq} resemble a correlated linear relationship. This observation is confirmed by the correlation in the test set visible in Table 4.4.

Table 4.3: Correlations and MSE for the model from fold 1 in the test set. Overall correlation using the whole test set was 0.647.

Test set			
Video	Time	Correlation	MSE
4	day	0.602	0.625
7	night	0.602	1.127
8	night	0.594	1.311
12	day	0.272	0.920
14	day	0.207	0.575

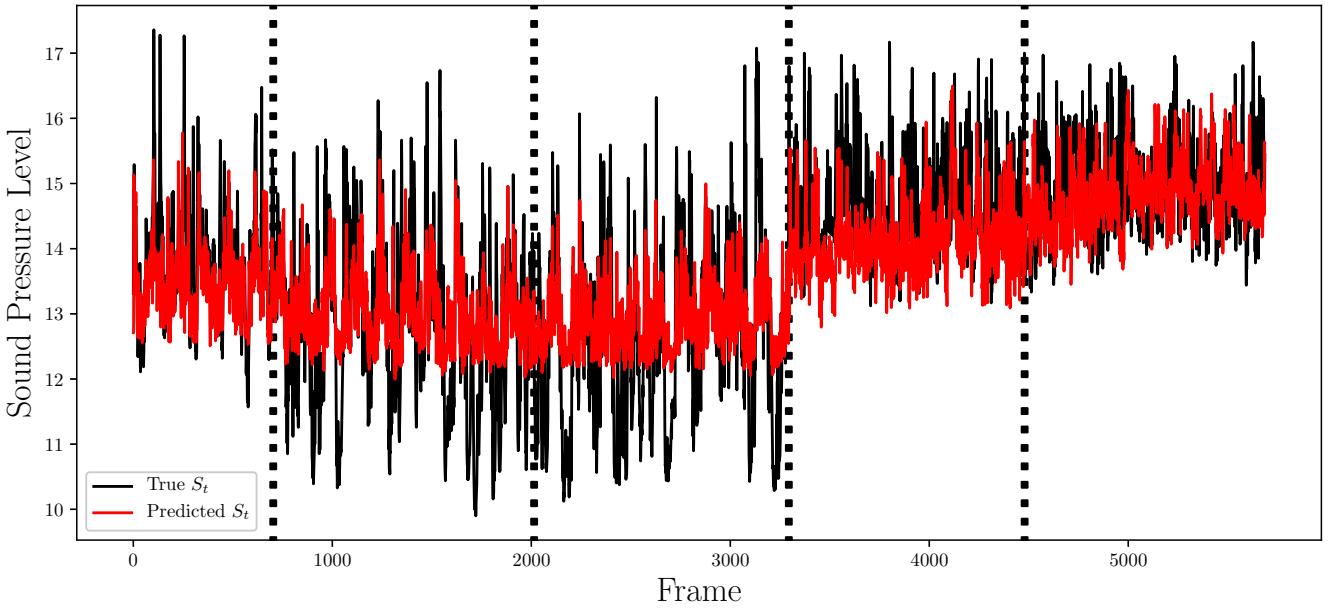


Figure 4.9: Predictions for model from fold 1 in the test set. True S_t are in black and predictions are in red. The dotted lines separate sections from each video. Videos are in order 4, 7, 8, 12 and 14.

Table 4.4: Correlations for model from fold 1 in the test set with L_{eq} with time interval equal to one minute converted. Correlation using the whole test set was 0.844.

Test set			
Video	Time	Correlation	MSE
4	day	0.630	8.186
7	night	0.651	5.103
8	night	0.715	6.785
12	day	0.255	9.182
14	day	0.124	2.739

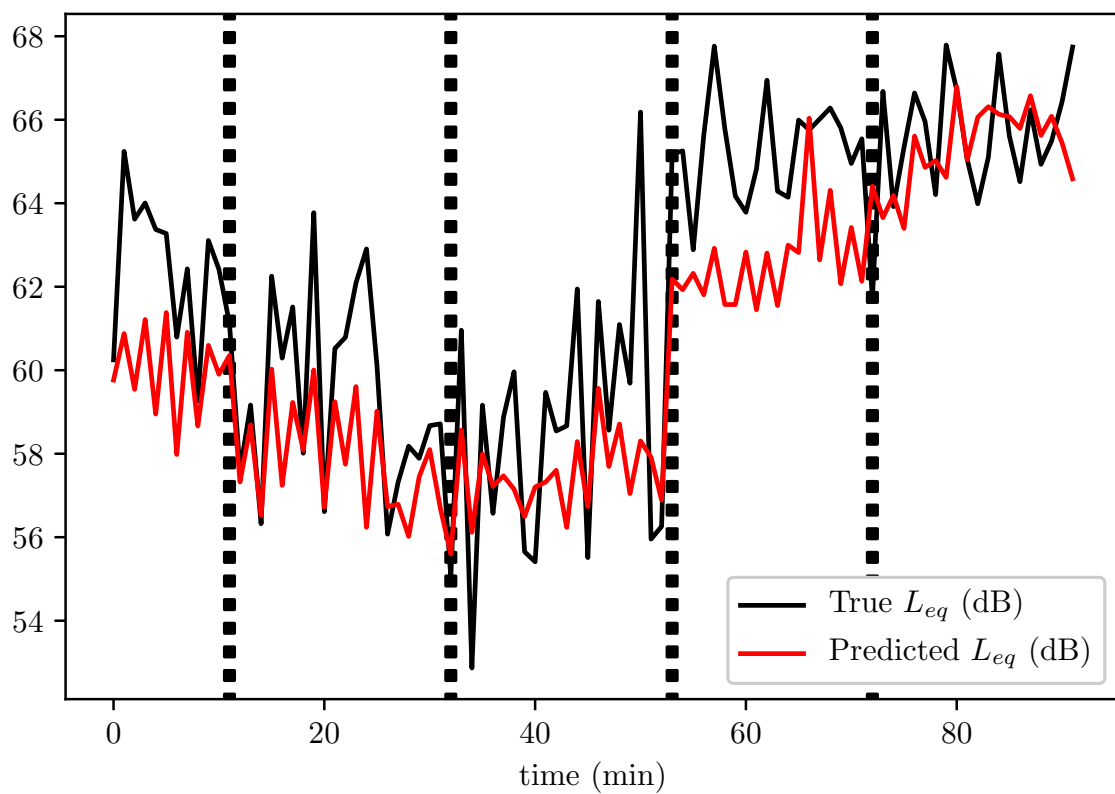


Figure 4.10: The dotted lines separate sections from each video. Videos are in order 4, 7, 8, 12 and 14.

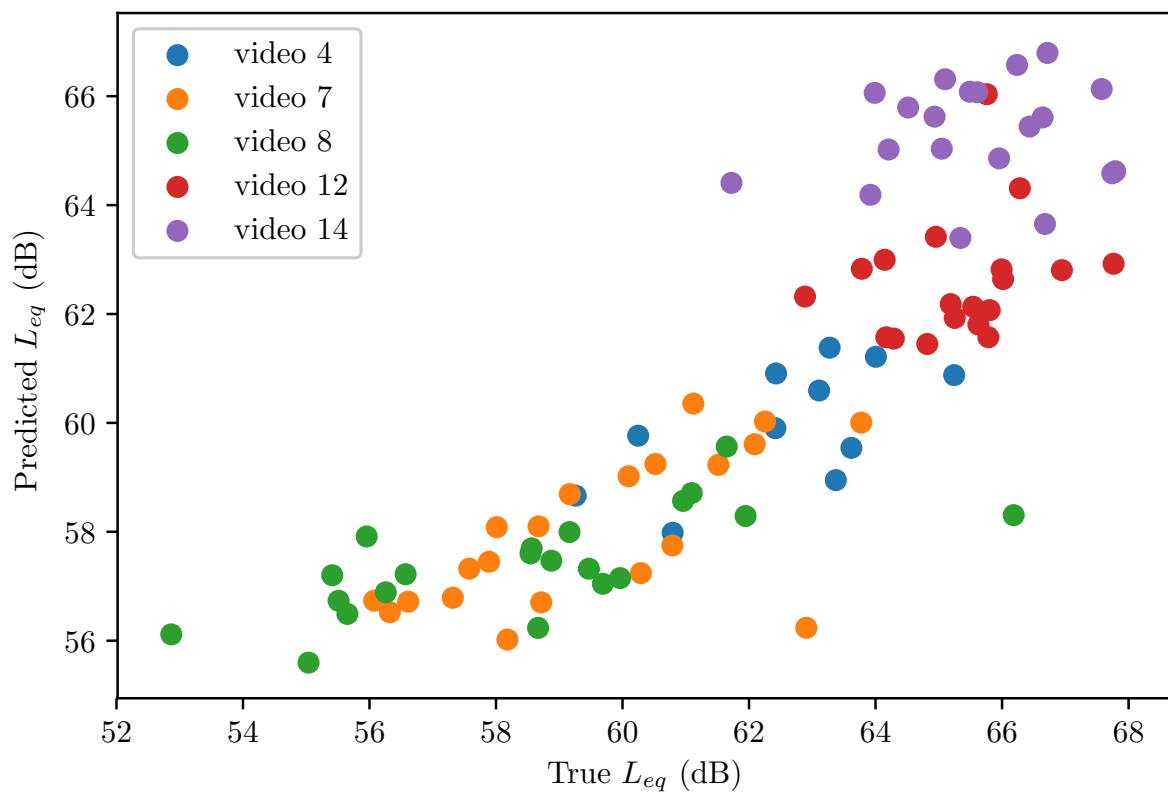


Figure 4.11: Relationship between the predicted L_{eq} and true L_{eq} (computed with $t_2 - t_1 = 1$ minute)

Chapter 5

Conclusion

Models 35 and 27 were able to follow trends in sound pressure level values. This occurred in spite of challenges in the dataset such as occasionally high S_t values without corresponding sound-generating sources within the frame. The performance of the two networks was compared and both predicted S_t values similarly. Model 27 in particular allowed the use of CAM to visualize image regions responsible for large network outputs. The CAM had to be temporally filtered so that changes that detect sound sources are observable. It was done by computing an online average CAM that was subtracted from the unprocessed CAM. This result was multiplied by a gain, upsampled to 240×240 by nearest neighbor interpolation and set as the green channel of the video frame. That produced a visualization where bright green spots represent the detection of a source. Finally, that helped us verify that the network predicts sound pressure by associating objects to S_t values. The training settings of model 35 were repeated for six folds and the best of those models was evaluated on the test set. This best model attained a correlation of approximately 0.6 in three of the videos and correlations of 0.272 and 0.207 in the other two. The low correlations in the two videos was associated with a traffic warden that constantly whistles: a characteristic not present in the training data of any of the videos used in the six folds. To try to predict S_t values in this situation, a new fold was created with three videos in the training set and two videos in the validation set. The training set contained one video with the traffic warden and the validation set contained one video with the traffic warden. In spite of that, the model was not capable of following trending S_t values in this situation. A final evaluation of the model was done with a longer term version of the sound pressure, a L_{eq} with a one minute time interval. This resulted in a correlation of 0.844 in L_{eq} estimation when all test videos are used. This indicates that estimation of longer term sound pressure levels is less sensitive to sporadic noise in the dataset.

5.1 Future Work

A possible development could be to use models capable of exploiting characteristics of audio and video signals. CNN+LSTM [27] models could be trained with knowledge of the temporal nature of the signals so that predictions would produce that type of behavior. For example, it is expected that a bus that is not moving would produce less sound pressure than a moving bus, so temporal information could be crucial for sound pressure prediction. In addition, a more general version of CAM could be applied to source detection. GradCAM [20] allows visualization of image regions responsible for the network outputs for more general models. In contrast to CAM, gradCAM does not require a network with GAP+FC after the last convolutional layer. In fact, gradCAM can localize image regions relevant to classification even in more complex models such as CNN+LSTM [20].

Another method to predict sound pressure levels could be to apply pretrained object detection networks such as YOLO [8] or Faster-RCNN [7] to initially count the number of buses, trucks, cars and motorbikes in the frame. The number of each vehicle could be represented as elements of a vector. This vector could then be used as input to a linear regression problem to predict sound pressure levels. This approach could also provide insights into the amount of sound pressure produced by each individual type of vehicle: vehicles associated with larger weights in the linear regression solution would be associated with larger predicted sound pressure level. More specifically, the linear regression weights would represent the amount of sound pressure expected per type of vehicle observed (plus bias).

This work could also be expanded by using online optimization methods in the learning scheme to detect audio sources. In the case of models one and three in Figure 3.4, if the parameters of the convolutional layers are not updated by the optimization algorithm, then the convolutional network acts as a function without trainable parameters applied to data. The training of parameters in further FC layers are blind to this function application. If f is the function applied by the convolutional part of a network with an image input \mathbf{x} , then:

$$\mathbf{x}' = [f(\mathbf{x}) \quad 1], \quad (5.1)$$

i.e. \mathbf{x}' is a function of the convolutional layers flattened to a vector and with a one concatenated at the end. In this situation, the optimization problem may be formulated as:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{N} \sum_n \|\boldsymbol{\theta}^t \mathbf{x}'_n - y_n\|_2^2, \quad (5.2)$$

where $\boldsymbol{\theta}$ represents all the FC parameters (including bias as the last element), y_n is the n -th target and \mathbf{x}_n is the n -th input. Equation (5.2) is a convex problem,

so parameters θ could be updated online with convergence guarantees by using upcoming frames and audio data. This could enhance the sound source detection with CAM by adaptively updating the FC parameters. A possible downside would be the susceptibility to inconsistent inputs such as when there are high sound pressure level samples measured without the corresponding object in the image itself. The model would learn from bad data and that would likely worsen prediction for new inputs. As a suggestion, the problem of inconsistent inputs could be mitigated by comparing the prediction of a parallel pretrained offline model to the current sound pressure level sample. If the prediction error of the parallel pretrained model is above some threshold, then it could be assumed that the audio source is not within the image itself and the update of the online model could cease until the error lowered. However, this would affect the long term capacity of the model to adapt. For example if the true sound pressure base level changes to an offset largely different from that of the data where the offline model was trained, any small audio change could trigger the threshold and cease adaptation. To overcome this issue, the offline and online models could be trained to predict not the absolute sound pressure level value, but a delta between the previous sound pressure level sample and the current sound pressure level sample.

Bibliography

- [1] STANSFELD, S. A., MATHESON, M. P. “Noise pollution: non-auditory effects on health”, *British Medical Bulletin*, v. 68, n. 1, pp. 243–257, dez. 2003. doi: 10.1093/bmb/ldg033. Disponível em: <<https://doi.org/10.1093/bmb/ldg033>>.
- [2] SØRENSEN, M., ANDERSEN, Z. J., NORDSBORG, R. B., et al. “Road Traffic Noise and Incident Myocardial Infarction: A Prospective Cohort Study”, *PLoS ONE*, v. 7, n. 6, pp. e39283, jun. 2012. doi: 10.1371/journal.pone.0039283. Disponível em: <<https://doi.org/10.1371/journal.pone.0039283>>.
- [3] W. EVANS, G., LEPORE, S. “Nonauditory Effects of Noise on Children: A Critical Review”, *Children’s Environments*, v. 10, pp. 31–51, 01 1993. doi: 10.2307/41515250.
- [4] “California Penal Code paragraph 632”. jan. 2017. Disponível em: <https://leginfo.legislature.ca.gov/faces/codes_displaySection.xhtml?lawCode=PEN§ionNum=632>. Accessed 23 April 2019.
- [5] KRIZHEVSKY, A., SUTSKEVER, I., E. HINTON, G. “ImageNet Classification with Deep Convolutional Neural Networks”, *Neural Information Processing Systems*, v. 25, 01 2012. doi: 10.1145/3065386.
- [6] HE, K., ZHANG, X., REN, S., et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun. 2016. doi: 10.1109/cvpr.2016.90. Disponível em: <<https://doi.org/10.1109/cvpr.2016.90>>.
- [7] REN, S., HE, K., GIRSHICK, R., et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *arXiv preprint arXiv:1506.01497*, 2015.
- [8] REDMON, J., FARHADI, A. “YOLOv3: An Incremental Improvement”, *arXiv*, 2018.

- [9] RONNEBERGER, O., FISCHER, P., BROX, T. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. 2015.
- [10] KUMAR, P., NIGAM, S., KUMAR, N. “Vehicular traffic noise modeling using artificial neural network approach”, *Transportation Research Part C: Emerging Technologies*, v. 40, pp. 111–122, mar. 2014. doi: 10.1016/j.trc.2014.01.006. Disponível em: <<https://doi.org/10.1016/j.trc.2014.01.006>>.
- [11] LEONARDI, G., CIRIANNI, F. “Artificial neural network for traffic noise modelling”, *ARPJ Journal of Engineering and Applied Sciences*, v. 10, 12 2015.
- [12] ISOLA, P., ZHU, J.-Y., ZHOU, T., et al. “Image-to-Image Translation with Conditional Adversarial Networks”. 2016.
- [13] GOODFELLOW, I. J., POUGET-ABADIE, J., MIRZA, M., et al. “Generative Adversarial Networks”. 2014.
- [14] SIMONYAN, K., ZISSERMAN, A. “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *CoRR*, v. abs/1409.1556, 2014.
- [15] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. *Deep Learning*. <http://www.deeplearningbook.org>, MIT Press, 2016.
- [16] CHOLLET, F., OTHERS. “Keras”. <https://keras.io>, 2015.
- [17] LIN, M., CHEN, Q., YAN, S. “Network In Network”. 2013.
- [18] HUANG, G., LIU, Z., VAN DER MAATEN, L., et al. “Densely Connected Convolutional Networks”. 2016.
- [19] ZHOU, B., KHOSLA, A., A., L., et al. “Learning Deep Features for Discriminative Localization.” *CVPR*, 2016.
- [20] SELVARAJU, R. R., COGSWELL, M., DAS, A., et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”. 2016.
- [21] RUSSAKOVSKY, O., DENG, J., SU, H., et al. “ImageNet Large Scale Visual Recognition Challenge”, *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, pp. 211–252, 2015. doi: 10.1007/s11263-015-0816-y.
- [22] GOYAL, P., DOLLAR, P., GIRSHICK, R., et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. 2017.

- [23] YOU, Y., ZHANG, Z., HSIEH, C.-J., et al. “ImageNet Training in Minutes”. 2017.
- [24] LIN, T.-Y., MAIRE, M., BELONGIE, S., et al. “Microsoft COCO: Common Objects in Context”. 2014.
- [25] “FFmpeg A complete, cross-platform solution to record, convert and stream audio and video.” <https://ffmpeg.org/>, 2019. Accessed: 2019-05-10.
- [26] “Python programming language”. <https://www.python.org/>, 2019. Accessed: 2019-05-10.
- [27] HOCHREITER, S., SCHMIDHUBER, J. “Long Short-Term Memory”, *Neural Computation*, v. 9, n. 8, pp. 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. Disponível em: <<https://doi.org/10.1162/neco.1997.9.8.1735>>.

Appendix A

Loss curves for the models in cross-validation

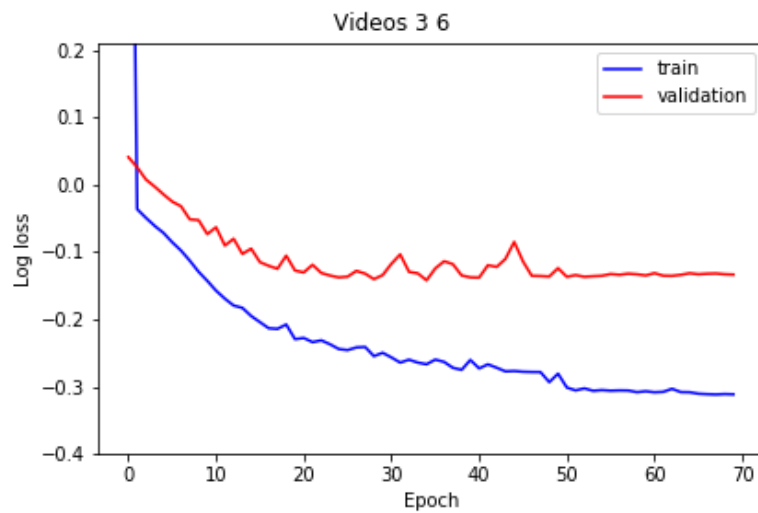


Figure A.1: Training and validation log loss curves in the fold 1.

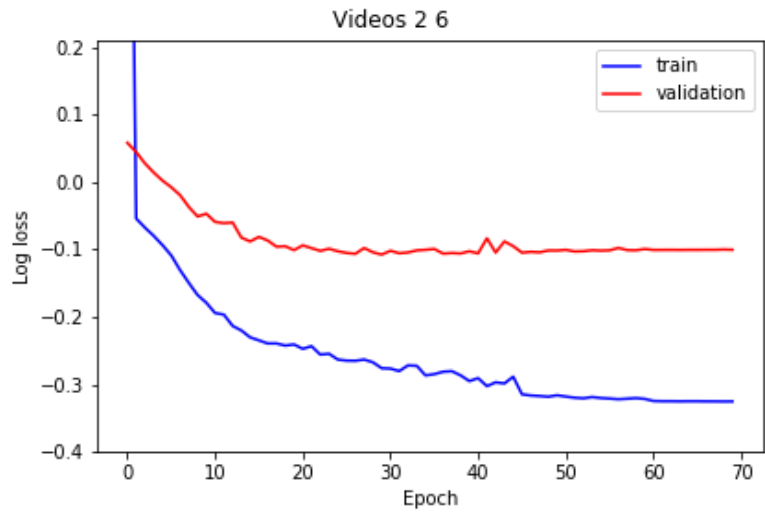


Figure A.2: Training and validation log loss curves in the fold 2.

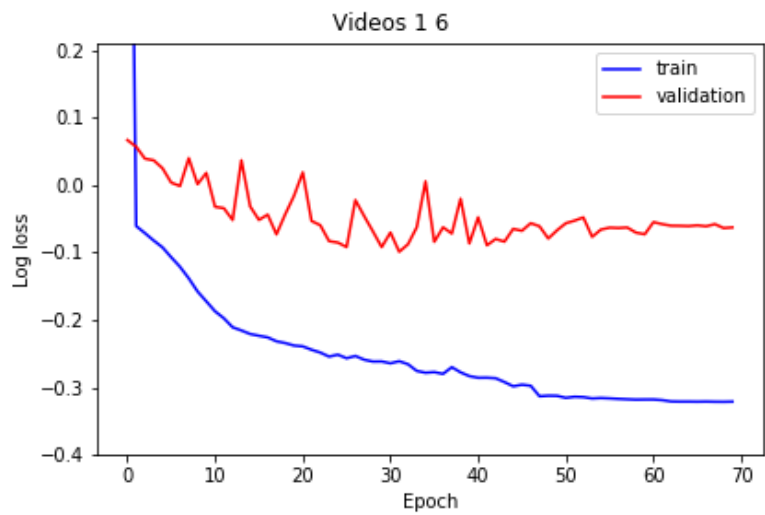


Figure A.3: Training and validation log loss curves in the fold 3.

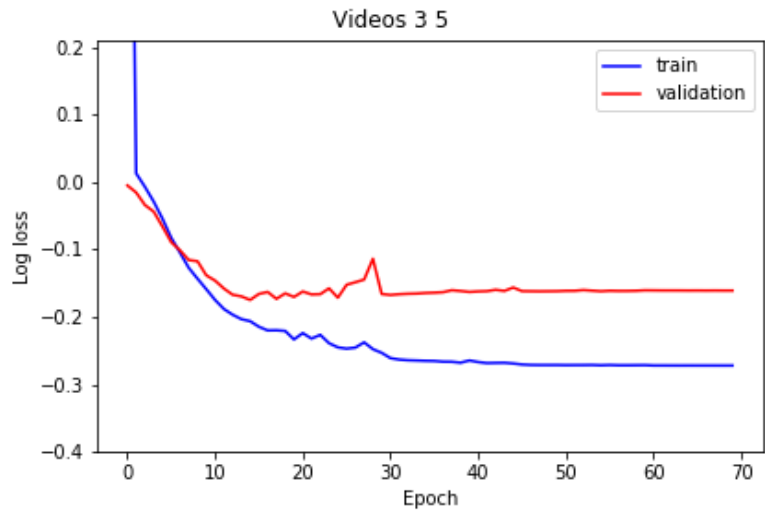


Figure A.4: Training and validation log loss curves in the fold 4.

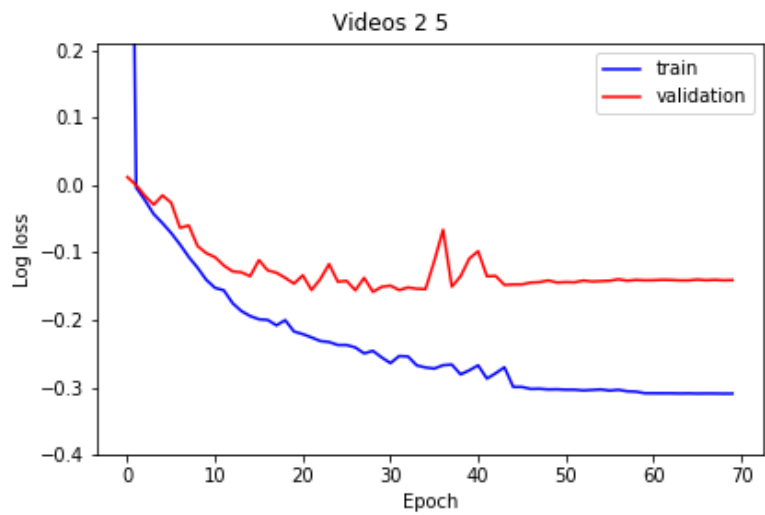


Figure A.5: Training and validation log loss curves in the fold 5.

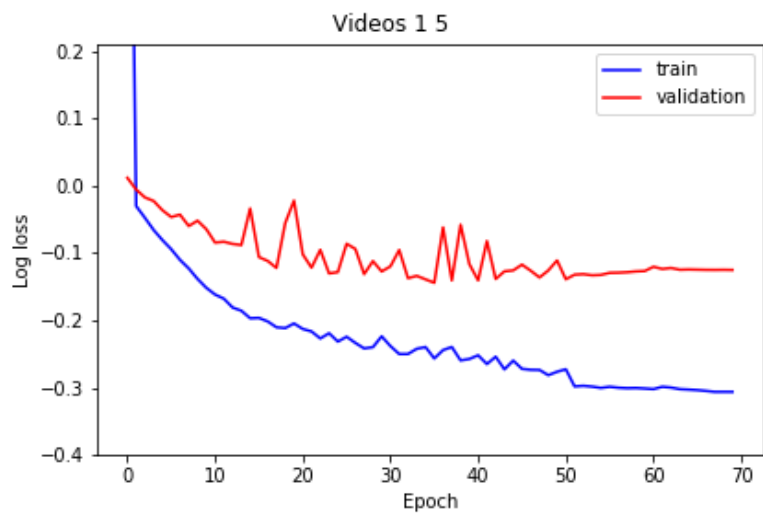


Figure A.6: Training and validation log loss curves in the fold 6.

Appendix B

Predictions of the models in cross-validation

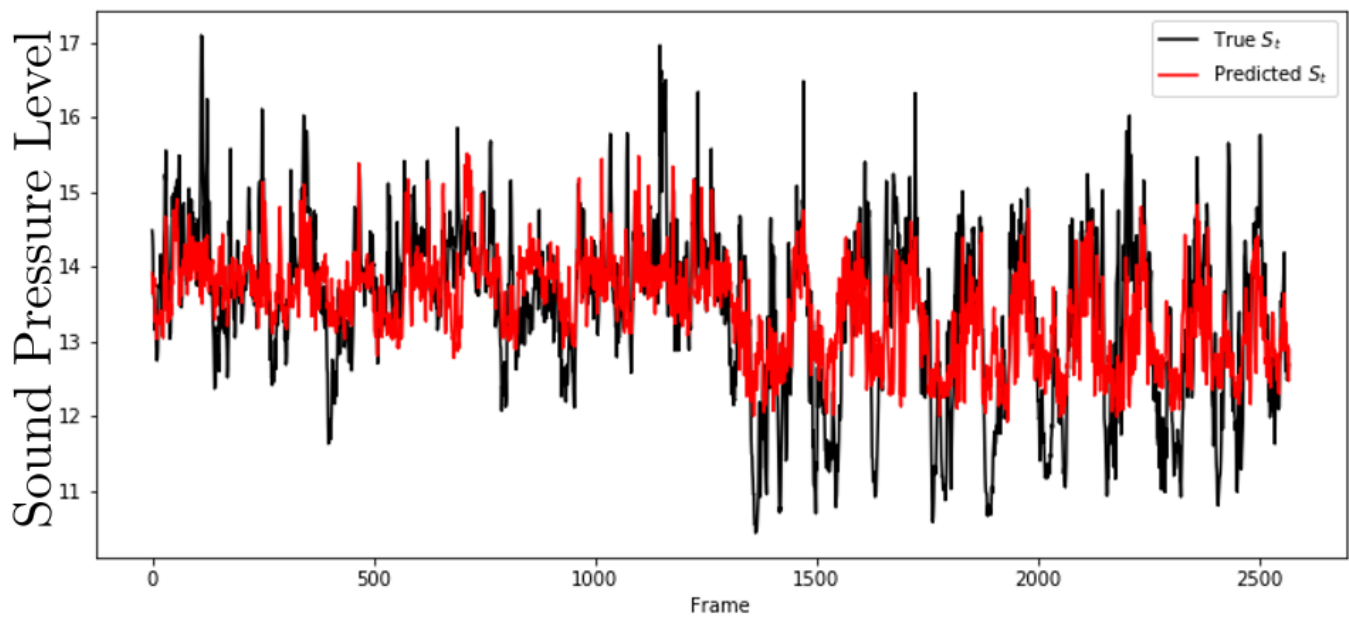


Figure B.1: Predictions of sound pressure level S_t in fold 1. The validation part of this fold contains frames from videos 3 and 6.

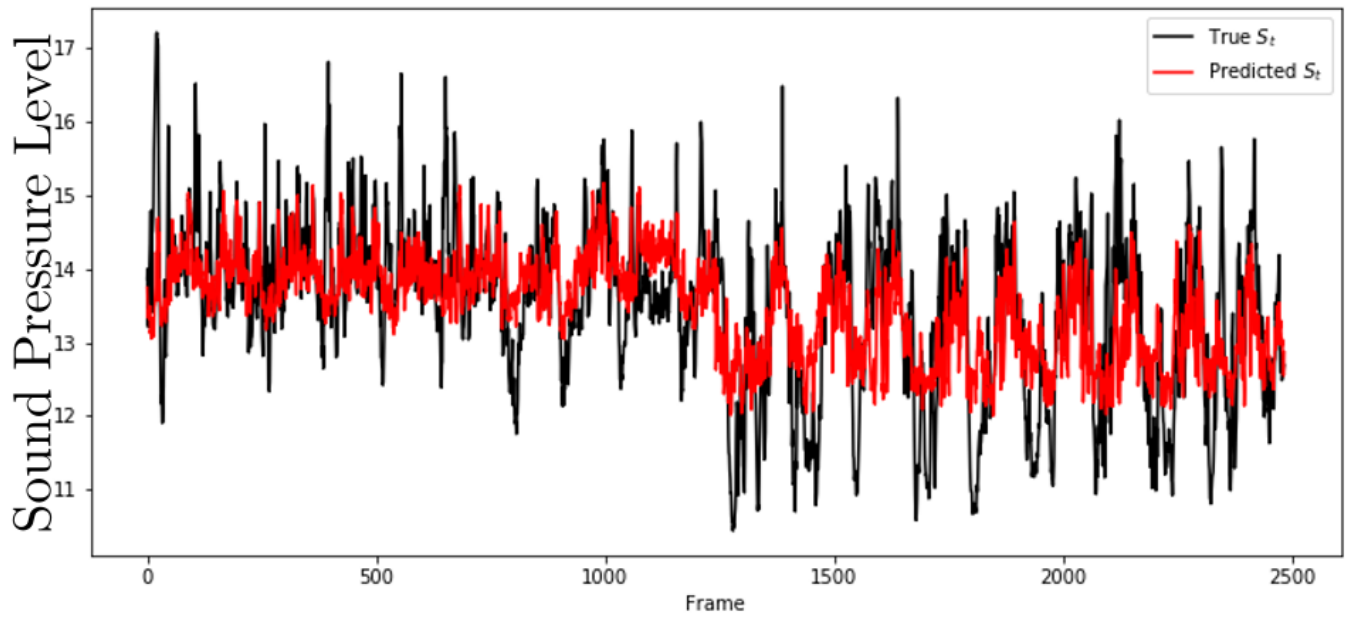


Figure B.2: Predictions of sound pressure level S_t in fold 2. The validation part of this fold contains frames from videos 2 and 6.

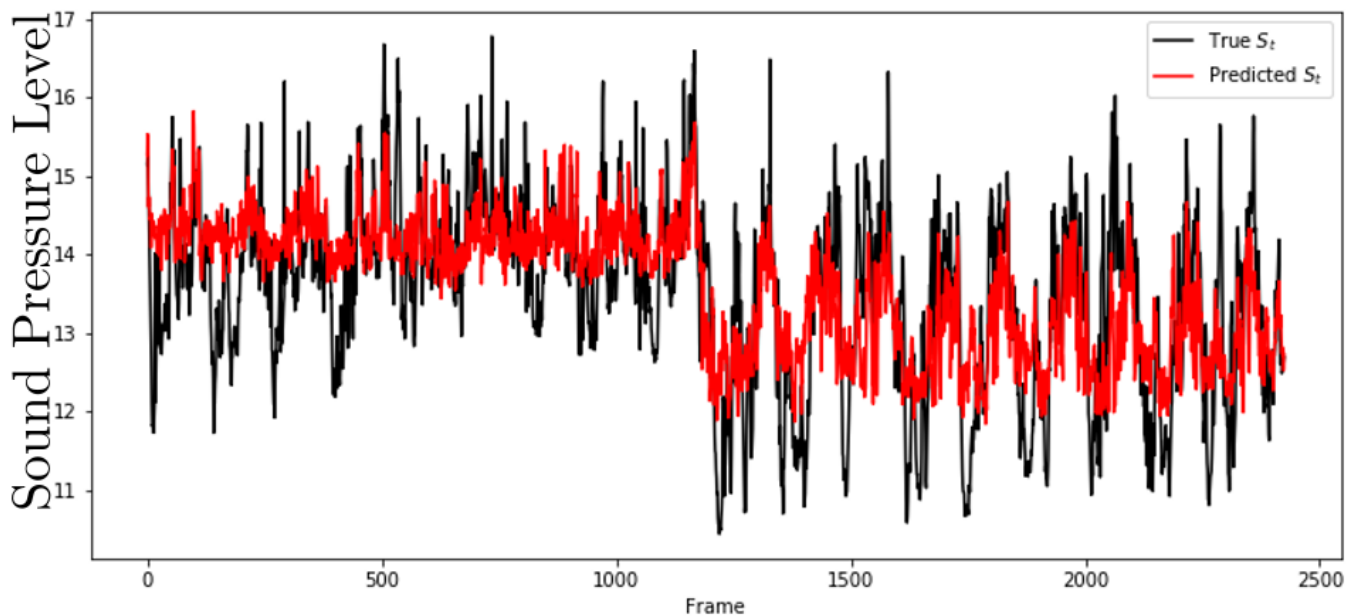


Figure B.3: Predictions of sound pressure level S_t in fold 3. The validation part of this fold contains frames from videos 1 and 6.

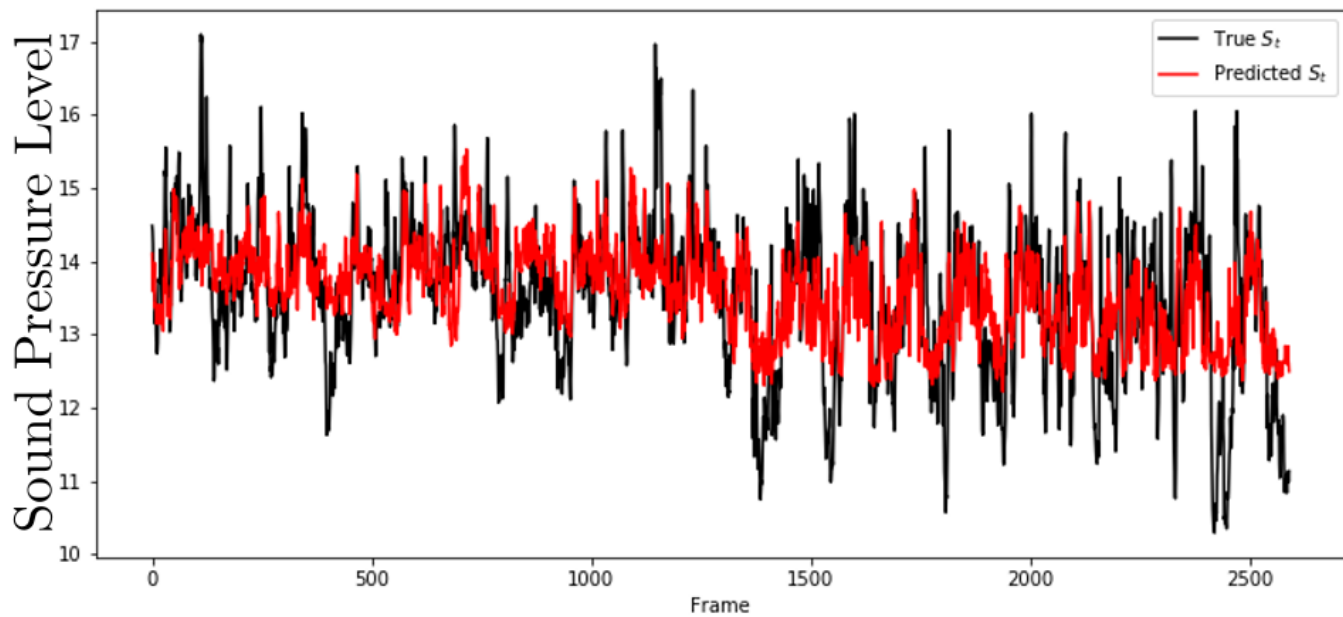


Figure B.4: Predictions of sound pressure level S_t in fold 4. The validation part of this fold contains frames from videos 3 and 5.

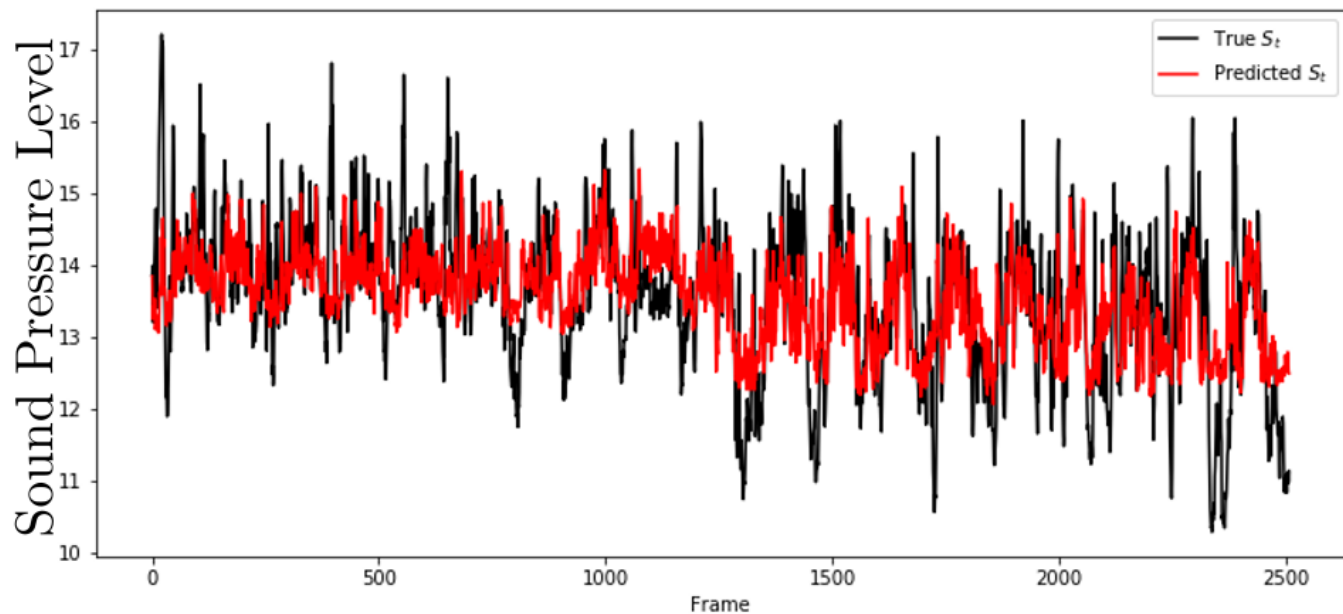


Figure B.5: Predictions of sound pressure level S_t in fold 5. The validation part of this fold contains frames from videos 2 and 5.

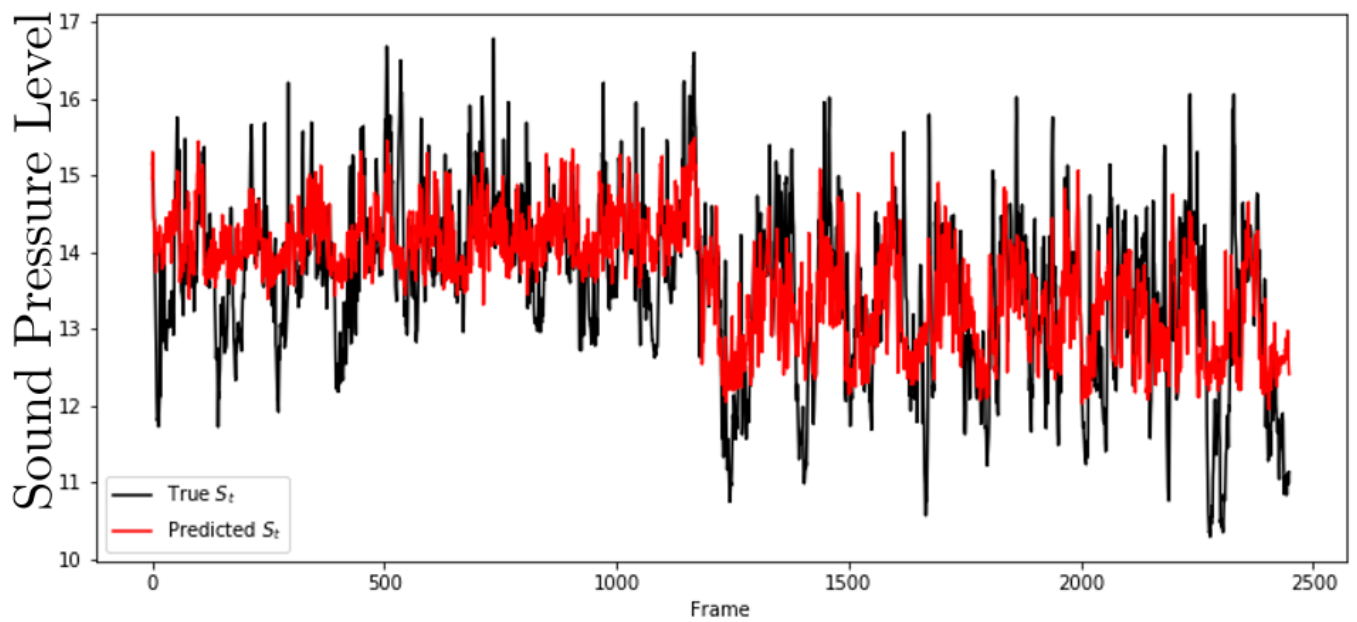


Figure B.6: Predictions of sound pressure level S_t in fold 6. The validation part of this fold contains frames from videos 1 and 5.