AN EXPLORATORY ASSESSMENT OF MULTISTREAM DEEP NEURAL NETWORK FUSION: DESIGN AND APPLICATIONS

ANA PAULA GONÇALVES SOARES DE ALMEIDA

DOCTORAL DISSERTATION IN MECHATRONIC SYSTEMS
DEPARTAMENT OF MECHANICAL ENGINEERING

SCHOOL OF TECHNOLOGY

UNIVERSITY OF BRASÍLIA

UNIVERSITY OF BRASÍLIA
SCHOOL OF TECHNOLOGY
DEPARTAMENT OF MECHANICAL ENGINEERING

AN EXPLORATORY ASSESSMENT OF MULTISTREAM DEEP NEURAL NETWORK FUSION: DESIGN AND APPLICATIONS

ANA PAULA GONÇALVES SOARES DE ALMEIDA

Supervisor: PROF. DR. FLÁVIO DE BARROS VIDAL, CIC/UNB/BRAZIL

DOCTORAL DISSERTATION IN MECHATRONIC SYSTEMS

BRASÍLIA-DF, 01 JULY 2022.

UNIVERSITY OF BRASÍLIA
SCHOOL OF TECHNOLOGY
DEPARTAMENT OF MECHANICAL ENGINEERING

AN EXPLORATORY ASSESSMENT OF MULTISTREAM DEEP NEURAL NETWORK FUSION: DESIGN AND APPLICATIONS

ANA PAULA GONÇALVES SOARES DE ALMEIDA

DOCTORAL DISSERTATION SUBMITTED TO THE DEPARTAMENT OF MECHANICAL ENGINEERING OF THE SCHOOL OF TECHNOLOGY OF THE UNIVERSITY OF BRASÍLIA, AS PART OF THE REQUIREMENTS FOR OBTAINING THE DOCTOR OF SCIENCE DEGREE IN MECHATRONIC SYSTEMS.

APPROVED BY:

BRASÍLIA, 01 JULY 2022.

INDEX CARD
ANA PAULA GONÇALVES SOARES DE ALMEIDA
An exploratory assessment of multistream deep neural network fusion: Design and applications
2022xv, 121p., 201x297 mm
(ENM/FT/UnB, Doctor of Science, Mechatronic Systems, 2022)
Doctoral Dissertation - University of Brasília
School of Technology - Department of Mechanical Engineering

BIBLIOGRAPHY

ANA PAULA GONÇALVES SOARES DE ALMEIDA (2022) An exploratory assessment of multistream deep neural network fusion: Design and applications. Doctoral Dissertation in Mechatronic Systems, Department of Mechanical Engineering, University of Brasília, Brasília, DF, 121p.

ASSIGNMENT OF THE RIGHT OF USE

AUTHOR: Ana Paula Gonçalves Soares de Almeida
TITLE: An exploratory assessment of multistream deep neural network fusion: Design and applications.
DEGREE: Doctor of Science YEAR: 2022

Ana Paula Gonçalves Soares de Almeida

Acknowledgements

I thank my parents, Rafael and Maria Gorete, for being the pillar of my entire life.

I thank my husband, André, for always supporting me in everything I dare to do.

I thank my advisor and friend Flávio for always believing in me and never leaving me helpless.

ABSTRACT

Machine-learning methods depend heavily on how well the selected feature extractor can represent the raw input data. Nowadays, we have more data and computational capacity to deal with it. With Convolutional Neural Networks, we have a network that is easier to train and generalizes much better than usual. However, a good amount of essential features are discarded in this process, even when using a powerful CNN. Multistream Convolutional Neural Networks can process more than one input using separate streams and are designed using any classical CNN architecture as a base. The use of M-CNNs generates more features and thus, improves the overall outcome. This work explored M-CNNs architectures and how the stream signals behave during the processing, coming up with a novel M-CNN cross-fusion strategy. The new module is first validated with a standard dataset, CIFAR-10, and compared with the corresponding networks (single-stream CNN and late fusion M-CNN). Early results on this scenario showed that our adapted model outperformed all the abovementioned models by at least 28% compared to all tested models. Expanding the test, we used the backbones of former state-of-the-art networks on image classification and additional datasets to investigate if the technique can put these designs back in the game. On the NORB dataset, we showed that we could increase accuracy up to 63.21% compared to basic M-CNNs structures. Varying our applications, the mAP@75 of the BDD100K multi-object detection and recognition dataset improved by 50.16% compared to its unadapted version, even when trained from scratch. The proposed fusion demonstrated robustness and stability, even when distractors were used as inputs. While our goal is to reuse previous state-of-the-art architectures with few modifications, we also expose the disadvantages of our explored strategy.

RESUMO

Os métodos de aprendizado de máquina dependem muito de quão bom o extrator de características selecionado pode representar os dados brutos de entrada. Atualmente, temos mais dados e capacidade computacional para lidar com isso. Com as Redes Neurais Convolucionais temos uma rede que é mais fácil de treinar e generaliza muito melhor do que o habitual. Há, no entanto, uma boa quantidade de características que são essenciais, mas são descartadas nesse processo, mesmo quando se utiliza uma CNN poderosa. As Redes Neurais Convolucionais Multistream podem processar mais de uma entrada usando fluxos separados e são projetadas usando qualquer arquitetura CNN clássica como base. O uso de M-CNNs gera mais informação de características e, assim, melhora o resultado geral. Este trabalho explorou arquiteturas M-CNNs e como os sinais de fluxo se comportam durante o processamento, chegando a uma nova estratégia de fusão cruzada de M-CNNs. O novo módulo é validado, inicialmente, com um conjunto de dados padrão, CIFAR-10, e comparado com as redes correspondentes (single-stream CNN e late fusion M-CNN). Os primeiros resultados neste cenário mostraram que nosso modelo adaptado superou todos os modelos mencionados acima em pelo menos 28% em comparação com todos os modelos testados. Expandindo o teste, usamos a base de antigas redes estado-da-arte na classificação de imagens e conjuntos de dados adicionais para investigar se a técnica pode colocar essas estruturas de volta ao jogo. No conjunto de dados NORB, mostramos que podemos aumentar a precisão em até $63,21\%$ quando comparado às estruturas básicas de M-CNNs. Variando nossas aplicações, o mAP@75 do conjunto de dados de detecção e reconhecimento de objetos BDD100K melhorou em $50,16\%$ em comparação com sua versão não adaptada, mesmo quando treinado do zero. A fusão proposta demonstrou robustez e estabilidade, mesmo quando distratores foram usados como entradas. Embora nosso objetivo seja reutilizar arquiteturas estado-da-arte anteriores com poucas modificações, também expomos as desvantagens de nossa estratégia explorada.

SUMMARY

LIST OF FIGURES

LIST OF TABLES

LIST OF TERMS AND ACRONYMS

ANN	Artificial Neural Network
CCM	Color Co-occurrence Method
CNNs	Convolutional Neural Networks
ELU	Exponential Linear Unit
FM	Fusion Module
fn	False Negative
fp	False Positive
GPU	Graphical Processing Units
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
L-CNN	Lattice Convolutional Neural Network
LSTM	Long Short-Term Memory
M-CNNs	Multistream Convolutional Neural Networks
mAP	Mean Average Precision
ReLU	Rectified Linear Units
RMSE	Root of the Mean Squared Error
SVM	Support Vector Machine
TDNN	Time-Delay Neural Network
tn	True Negative
tp	True Positive

LIST OF SYMBOLS

J	Objective function.
P	Perceptron.
a	Parameter of inclination.
g	Ground truth of the samples.
g'	Prediction of the samples.
ip	Perceptron input.
wg	Perceptron weights.
C	Convolutional layer.
F(s)	Fusion function.
L	Layer.
OP	Fusion mathematical operation.
d	Depth.
h	Height.
s	Signal.
w	Width.
x	Input width coordinate.
y	Input height coordinate.
$\boldsymbol{I}$	Image input.
$\boldsymbol{K}$	Kernel function.
$\boldsymbol{O}$	Feature map.

Chapter 1

Introduction

Our society has always yearned for automated machines. It has been a topic of philosophy, general literature, movies. Since the introduction of the idea of neural networks as computing machines [13] and Rosenblatt's Perceptron [14] — the first model for supervised learning [15] —, we were able to solve mathematical-based problems that are naturally difficult for human beings [1]. However, we still have a long way to go when talking about easy and intuitive tasks for humans but challenging for machines, such as face recognition.

Machine-learning methods depend heavily on how well the selected feature extractor can represent the raw input data. And these extractors require a significant amount of knowledge to be constructed [16]. On the other hand, deep learning is a different approach to artificial intelligence. It is capable of learning multiple levels of representation, starting with the raw input, by using non-linear modules [16]. These higher layers of representation amplify the essential features of the data and suppress irrelevant variations [16].

Convolutional Neural Networks (CNNs) are a specialized type of deep, feedforward networks, considered easier to train and with better generalization than networks with full connectivity between adjacent layers [16]. CNNs have achieved impressive results in different tasks of computer vision, such as image classification [3, 17, 18], segmentation [19, 20], object detection [21, 22, 23], and action recognition [24, 6, 7, 25].

Especially in action recognition tasks, a novel approach of fusing CNNs was proposed by Karpathy (2014) [6]. Multistream (or multichannel) Convolutional Neural Networks (M-CNNs) are networks that have two or more streams, aiming for better performance with less training time. The objective of augmenting the number of streams is to extend the connectivity of a CNN in the time domain to take advantage of local Spatio-temporal information [6]. Applying this multistream technique to static problems, as image classification, also outperforms single-channel results, considering that all the streams have contributive data to the network [11, 12].

1.1 Motivation

Nowadays, we have more data and computational capacity to deal with it. Consequently, we can also observe an increasing the complexity of the solutions, even when they were not necessary [26]. Unnecessary layer stack and ramification needs computing power that directly affects the environment, and the most important point: larger and more elaborate networks have a higher degradation state throughout the training process.

Considering the number of important but discarded features in a deep neural network process, we desire to represent this data in a better way. Thus, we implement M-CNNs for applications besides video classification to evaluate if this kind of network setting is better for the general performance of the analyzed problem and where there is room for improvement.

Furthermore, we propose a novel fusion module capable of improving M-CNN results by performing operations with the multiple streams and replacing its connections, observing how the fusion process can optimize the overall network. Different kinds of applications will serve as evaluation material for this new model.

In addition to the module, we address the following hypothesis **H1**:

Hypothesis 1 (H1): *The proposed new module is a possible approach to recycle architectures that are no longer used — because more complex structures are now state-of-the-art —, making the so-called obsolete artificial neural networks competitive again.*

With the motivation and the hypothesis, we define the subsequent goals.

1.2 Goals

This work aims to achieve the following contributions, using an exploratory and sequential methodology approach, divided into Primary and Secondary goals.

1.2.1 Primary Goal

Design, improve and explore current models of CNNs and M-CNNs to propose a novel structural module that handles all the intermediate processes of M-CNNs.

1.2.2 Secondaries Goals

To achieve our primary goal, we have to set intermediary objectives, being them:

- Develop different M-CNN applications to compare its results to traditional CNN models;

- Propose novel M-CNN architectures focusing on improvements based on structural modifications;
- Explore signal processing boosting techniques inside the structure of M-CNNs;
- Design a general boosting module to adapt classical CNNs;
- Test different applications with the novel module, using same-domain and cross-domain streams;
- Explore how the novel module performs with simpler networks.

1.3 Main Contributions

Knowing that M-CNNs have been developed, applied, and used in many situations and applications [6, 12, 27, 28, 29, 30, 31, 32, 33], we implemented different M-CNN architectures to discover the feasibility of using this type of network. These implementations were the first contributions of this work [11, 12].

While our overall goal is to lessen the complexities, it's important to note that these M-CNNs duplicate the original networks. That is, despite the shorter training time, we still require a certain computational power. Based on the discoveries of the papers mentioned above, we noticed that works that address applications issues to an M-CNN approach do not focus on the fusion stage.

Thus, we propose in this manuscript a novel module based on a new cross-fusion method applied for all available models of M-CNNs. This new proposed cross-fusion method is focused on observed features, used by many of traditional M-CNNs models developed in all known approaches.

By adapting simpler and traditional structures, we expect this nouveau strategy to possibly outperform the classical CNN networks and encourage the reuse of out-of-date networks. With extensive tests within different application scenarios, we managed to identify the strong and weak points of this approach.

1.4 Manuscript Organization

This work is divided as follows: Chapter 2 presents a theoretical background to understand further chapters. Chapter 3 details CNNs and their variant, M-CNNs. It also introduces related work, state-of-the-art, and important CNN and M-CNNs architectures. Furthermore, it contains state-of-the-art applications using single and multiple stream data. Chapter 4 presents the methodology. From Chapter 5 in Sections 5.1 up to 5.4, studied and implemented M-CNNs applications are showed as contributions and filling gaps, and Chapter 6 a new proposed topology is presented describing the main concept of the fusion strategy and the module itself. Finally, Chapter 7 contains the conclusion and future works.

Chapter 2

Theoretical Background

This chapter briefly describes an overview of basic and main concepts of how Convolutional Neural Networks work.

2.1 Basic concepts

The perceptron is a probabilistic model introduced by Rosenblatt (1958) [34] defined by Equation 2.1, where P is the binary answer of a n-dimensional perceptron, ip is the input, wg are the weights, and ζ is the threshold.

$$P = \begin{cases} 0 & if \sum_n wg_n ip_n \leq \zeta \\ 1 & if \sum_n wg_n ip_n > \zeta \end{cases} . \tag{2.1}$$

This is the simplest model of neural networks used to classify linearly separable patterns; if it is built around a single neuron, the classification is limited to a binary output [15].

Multilayer perceptrons are a generalization of the single-layer perceptron previously defined. According to Haykin (1994) [15], a multilayer perceptron can be described as a neural network with one or more hidden layers, where these hidden neurons act as feature detectors. As the learning process progresses across the multilayer perceptron, the hidden neurons gradually discover the salient features that characterize the training data by performing a non-linear transformation on the input data into a feature space. In this new space, the classes of interest in a pattern-classification task, for example, may be more easily separated from each other than could be the case in the original input data space.

2.2 Convolutional Neural Networks

CNNs are a special class of multilayer perceptrons that are designed to process multiple arrays, such as images, with a high degree of invariance to translation, scaling, skewing, and other forms of distortion [15, 16].

In machine learning applications, the input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm [1] — these multidimensional arrays are called tensors. Considering the input as an image $\boldsymbol{I}$ with (x, y) coordinates and a two-dimensional kernel function $\boldsymbol{K}$, a feature map $\boldsymbol{O}$ can be achieved by calculating a discrete convolution operation described in Equation 2.2.

$$\boldsymbol{O}(x, y) = (\boldsymbol{K} * \boldsymbol{I})(x, y). \tag{2.2}$$

Since each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but in the finite set of points for which we store the values. This means that we can implement the infinite summation as a summation over a limited number of array elements [1], as stated in Equation 2.3.

$$\boldsymbol{O}(x, y) = \sum_m \sum_n \boldsymbol{I}(x - m, y - n)\boldsymbol{K}(m, n). \tag{2.3}$$

Usually, CNNs are structured as a series of blocks, where the first block is composed of convolutional and pooling layers [16]. The convolutional layer performs several convolutions in parallel to produce a set of linear activations. Each linear activation is run through a nonlinear function in the second stage. Then, the pooling layer modifies the layer's output further with a pooling function. A pooling function replaces the output of the net at a specific location with a summary statistic of the nearby outputs [1]. These components are summarized in Figure 2.1.

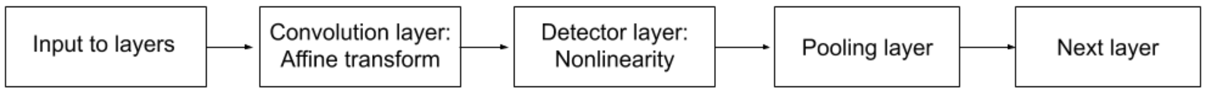

Input to layers
Convolution layer: Affine transform
Detector layer: Nonlinearity
Pooling layer
Next layer

Figure 2.1: Typical components of the first block of a convolutional neural network. Adapted from Goodfellow *et al.* (2016) [1].

The number of feature maps is increased with the successive blocks alternating between convolution and pooling. At the same time, the spatial resolution is reduced, compared with the corresponding previous block [15] — the following Section details other CNN common layers, as regularization and activations.

2.2.1 Additional Layers

2.2.1.1 Activations

Considering a neuron as a fundamental unity of a neural network processing, we describe one of its basic elements: an activation function. Haykin (1994) [15] defines the activation function as a restriction of the output amplitude of a neuron, having an output interval generally of a unit interval $[0, 1]$ or $[-1, 1]$.

The sigmoid function is the most common activation function. It has the shape of a *s*, and one example of it is the logistic function, defined as

$$sigmoid(x) = \frac{1}{1 + exp(-ax)}, \tag{2.4}$$

where a is the parameter of inclination.

Since the sigmoid function is used to represent a probability distribution over a binary variable, we need to extend this representation to *n* possible values. This generalization is the softmax function — defined in Equation 2.5 —, most used as the output of a classifier [1].

$$softmax(x) = \frac{exp(x)}{\sum_n exp(x_n)}. \tag{2.5}$$

However, the recommendation for modern neural networks is a rectified linear unit (ReLU) because being a nearly linear function, they preserve many of the properties that make linear models generalize well [1]. Additionally, a ReLU is a simple maximum function, defined as $ReLU(x) = max\{0, x\}$ [35].

2.2.1.2 Optimization

According to Salas *et al.* (2019) [26], the minimization of the loss function is the quantification of the concept of learning. Using the loss function as a parameter to the optimization process, several algorithms can be applied to diminish this loss. Gradient descent and its variations are the favorite methods for this task.

Nonetheless, the search space for the best parameters during optimization can be extensive. It can become a problem of overfitting, where a model generalizes too well to the input but is not able to work with new data.

2.2.1.3 Regularization

The theory of regularization helps to reduce overfitting at the expense of increased training error. Many regularization methods limits the capacity of models by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J, denoted by $J(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$, where $\alpha \in [\theta, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term relative to the standard objective function J.

A L2 regularization, or weight decay, adds an extra parameter to a cost function: a regularization term $\Omega(\theta) = \frac{1}{2} \|w\|_2^2$. The effect of regularization is to make the network prefer to learn small weights, all other things being equal. Large weights will only be allowed if they considerably improve the first part of the cost function [1].

The penalty term is substituted for a constraint using the standard stochastic gradient descent. Instead of penalizing the L2 norm of the whole weight vector, an upper bound on the L2 norm of the incoming weight vector for each hidden unit is set. When an update breaches this restriction, the weights of the hidden unit are renormalized by division. As stated in Hinton *et al.* (2012) [36], this makes it possible to start with a considerable learning rate that decays during learning, thus allowing a far more thorough search of the weight-space than methods that start with small weights and use a small learning rate.

Another regularization technique is called dropout. Dealing directly with structure changes, some hidden units are randomly dropped during training, preventing co-adaptation and forcing each hidden unit more robust and drive it towards creating useful features on its own without relying on other hidden units to correct its mistakes [36].

2.2.2 Time-review

The use of CNNs improved the state-of-the-art in visual object recognition [3], object detection [37], image segmentation [19] and many other domains [38, 39]. Figure 2.2 shows a compilation of terms often associated with the *Convolutional Neural Networks*, where the larger blobs represent a term that is constantly used. It can be noted that application terms, such as recognition and classification, appear in bigger blobs and great frequency. A time-review is presented in the next Chapter, from most to less recent of the main literature works.

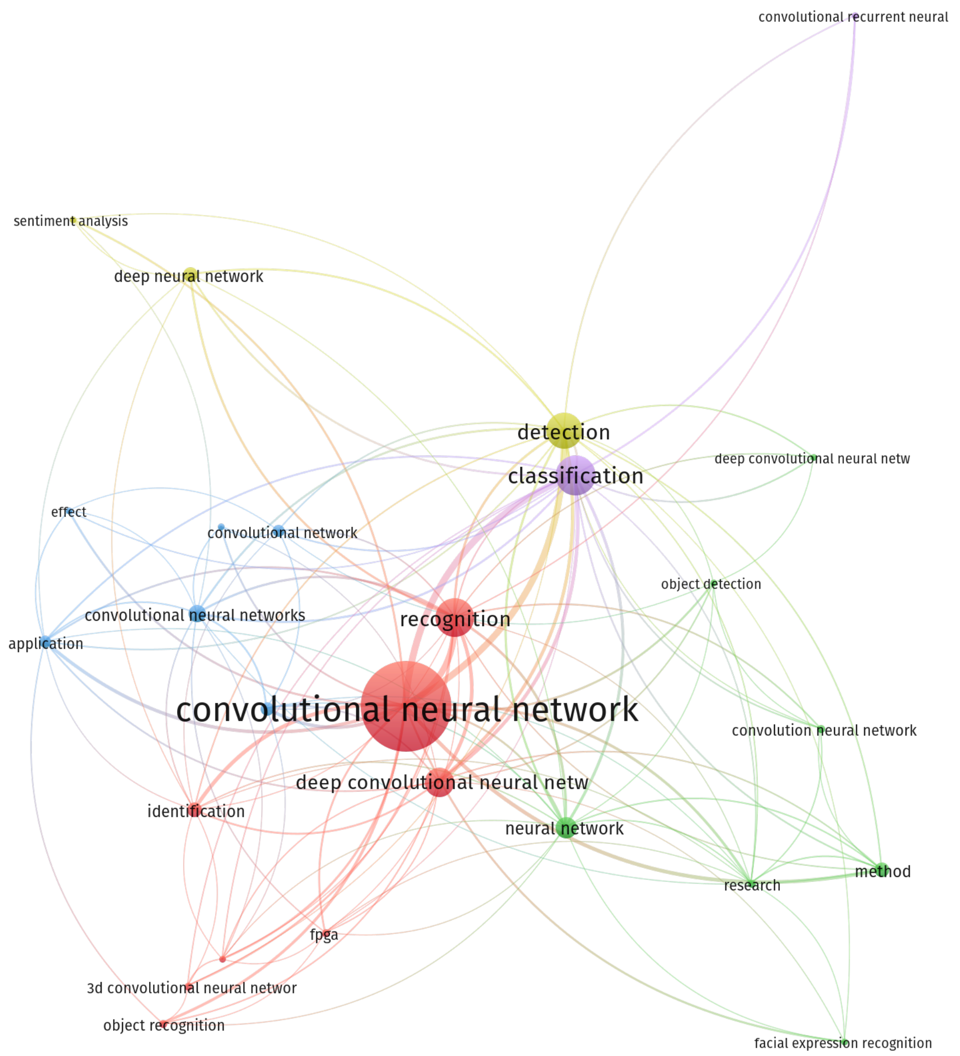

convolutional recurrent neural
sentiment analysis
deep neural network
detection
classification
deep convolutional neural netw
effect
convolutional network
object detection
convolutional neural networks
recognition
application
convolutional neural network
convolution neural network
deep convolutional neural netw
identification
neural network
research
method
fpga
3d convolutional neural networ
object recognition
facial expression recognition

Figure 2.2: Terms that are often associated with CNNs.

Chapter 3

Related Work

In 1980, a self-organizing neural network model named *Neocognitron* was proposed by Fukushima (1980) [40] to recognize patterns regardless of the position of the stimulus patterns. The multilayered structure of Neocognitron is based on the visual nervous system of the vertebrate. After the self-organization, the network acquires a design similar to the hierarchy model of the optical nervous system proposed by Hubel *et al.* (1962) [41], exploiting the notion of "simple" and "complex" cells. It can be considered the very first CNN, but it did not have an end-to-end supervised-learning algorithm such as backpropagation [16].

Years later, in 1989, Waibel *et al.* (1989) [42] proposed an approach to phoneme recognition named Time-Delay Neural Network (TDNN), a primitive 1D CNN. A TDNN is a multilayered feedforward neural network that can represent relationships between events in time, not requiring temporal alignment of the labels. The features learned by the network are insensitive to shifts in time. The first paper to use CNNs trained by backpropagation for the task of classifying low-resolution images of handwritten digits were proposed by LeCun *et al.* (1990) [43] and was a breakthrough in the field, with the presentation of LeNet-1. The LeNet-1 is a multi-layer network with convolutions with small size kernels. The followed stages are a squashing function and an additional layer that performs a local averaging and subsampling, reducing the resolution of the generated feature map. In 1998, LeCun [2] published a paper reviewing various methods applied to handwritten character recognition and compared them, showing that CNNs outperform all other techniques. It also proposes an evolution to the original LeNet-1 to LeNet-5 model.

According to Girshick *et al.* (2014) [44], CNNs saw heavy use in the 1990s [43, 2], but then fell out of fashion with the rise of Support Vector Machines (SVMs). In 2012, however, Krizhevsky *et al.* (2012) [3] revived the interest in CNNs by showing substantially higher image classification accuracy — 84.70% — on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [45], almost halve the error rate for object recognition at the time. Likewise, the increasing hardware processing capability and popularization of Graphical Processing Units (GPUs) were additional assets to the works.

The impact of the convolutional network called AlexNet and the revival of the CNN topic can also be seen in the graphic presented in Figure 3.1 of publications by year, since 2009, from three of the most prominent digital research libraries of the Artificial Intelligence field: ACM [46], IEEE Xplore [47], and ScienceDirect [48]. Correlate terms as *convolutional neural network, ConvNet* and *deep learning* were also used to highlight the effects of such event. It is also interesting to note in Figure 3.1 that the sharp drop in the amount of work on CNNs is related to the influence of the Sars-Cov-2 pandemic.

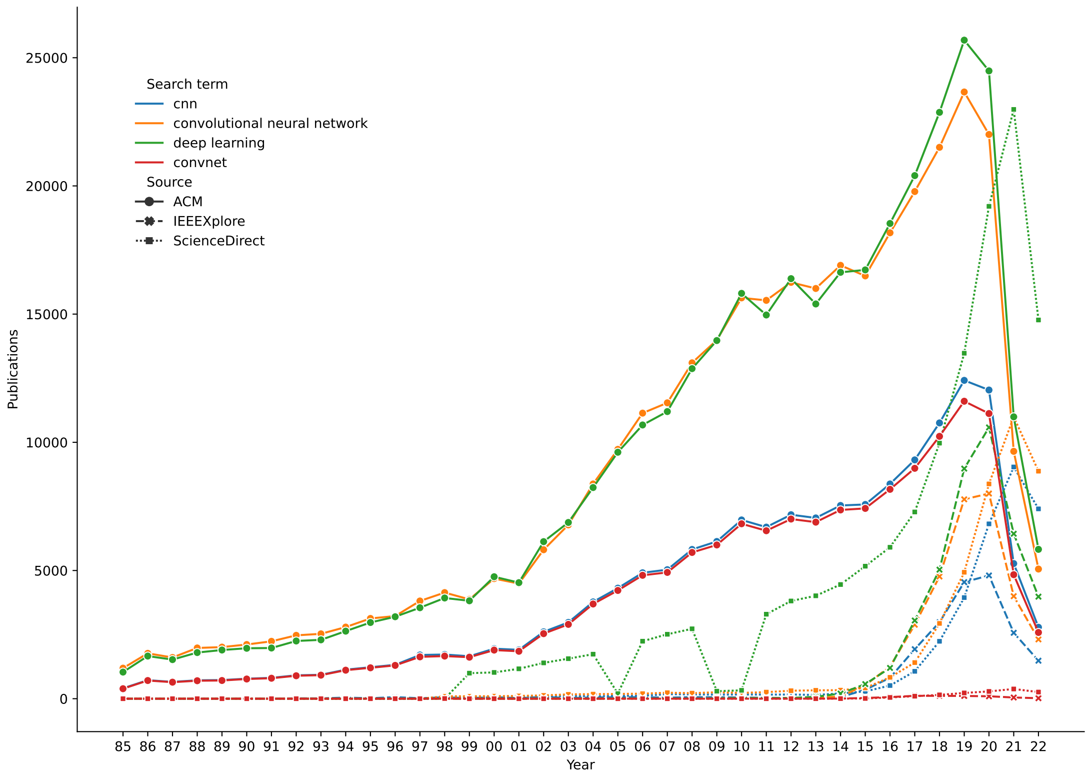


Figure 3.1: Occurrences of the terms: *CNN, convolutional neural network, ConvNet* and *deep learning* in 3 different repositories (ACM, Elsevier/ScienceDirect and IEEEXplore).

The architecture of AlexNet contains eight learned layers, five convolutional and three fully-connected [3]. It uses techniques like Rectified Linear Units (ReLUs) [35] as an activation function, speeding up the training; and dropout layers [36] to reduce overfitting.

The ImageNet challenge proved to be one of the primary sources of novelties in CNNs architecture. Between 2014 and 2015, three different CNN models were proposed and remain popular. Furthermore, the increasing number of layers characterizes this evolution. In Simonyan *et al.* (2014) [17], there is an evaluation of networks of increasing depth using an architecture with small convolution filters, showing that improvement can be achieved by increasing the weight layers depth. The proposed networks are VGG-16 and VGG-19.

In 2015, GoogLenet — or Inception v1 — was proposed by Szegedy *et al.* (2015) [4]. This 22-layers architecture values the use of computing resources inside the network. They increased the depth and width of the network while keeping the computational budget constant.

With more layers than the previous, ResNet [18] showed up as a residual learning framework that improved the training of deeper networks. According to He *et al.* (2015) [18], with the network depth increasing, accuracy gets saturated and then degrades rapidly. This degradation is not caused by overfitting, and adding more layers to a deep model leads to higher training error. They explicitly let stacked layers fit a residual mapping to address this issue.

Besides state-of-the-art architectures, developers have also worked in a wide gamma of CNN applications. In Wang *et al.* (2019) [38], it is proposed a method of learning from naturally distributed data and optimizing the classification accuracy over a balanced test set. This method maps an image to a feature space that visual concepts can relate to each other based on a learned metric that respects the closed-world classification while acknowledging the novelty of the open world, featuring image recognition is performed in large-scale databases.

Furthermore, dense 3D face decoding at a high frame rate was also contemplated. Zhou *et al.* (2009) [49] present a non-linear statistical model that represents facial texture and shape variations by learning joint texture and shape auto-encoders using direct mesh convolutions.

Although most of the applications are oriented toward images [50] — primarily image classification and object recognition —, plentiful models are using other data sources. For instance, Wang *et al.* (2017) [39] propose a wavelet-based ensemble approach for probabilistic wind power forecasting. Other examples include 3D object classification from point clouds [51], plant diseases recognition [12], and recognition of human activity in indoor environments [52].

Meanwhile, Karpathy *et al.* (2014) [6] extended the image recognition good results to video classification, developing a network structure with two different inputs to the classical CNN. This multistream technique will be explored in the following Section.

3.1 Main CNNs Architectures and Models

This Subsection describes the most relevant and used CNNs architectures in chronological order.

3.1.1 LeNet-5

The late 90's LeNet-5 [2] presented in Figure 3.2 is a 7-layer CNN that receives as input an approximately size-normalized and centered 32x32 pixel image. This input size was meant to be larger than the dataset images to center the potential distinctive features of the data, such as corners.

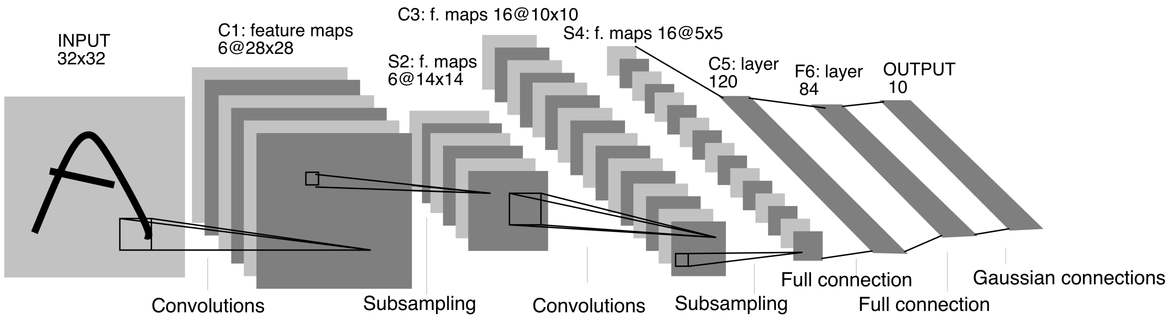


Figure 3.2: LeNet-5 architecture proposed by LeCun *et al.* (1998) [2]. Figure originally found in [2]. © 1998 IEEE.

The first convolutional layer, C1, has six 28x28 sized feature maps. Each unit in each feature map is connected to a 5x5 neighborhood in the input. Afterward, there is a subsampling layer with six 14x14 sized feature maps. These feature maps' units are connected to a 2x2 neighborhood in the corresponding feature map in C1.

These connections are sustained until the third convolutional layer. This behavior causes a break of symmetry in the network, so different feature maps are forced to extract different features [2].

3.1.2 AlexNet

The AlexNet architecture [3] showed in Figure 3.3 has eight layers. With an input size of $224 \times 224 \times 3$, the first convolutional layer filters the image with 96 kernels of size $11 \times 11 \times 3$ with a distance between the receptive field centers neighboring neurons in a kernel map of 4 pixels. The output of the first convolutional layer becomes the input of the second convolutional layer. The third, fourth, and fifth convolutional layers are connected without pooling layers. The first two fully-connected layers use a dropout [36] technique to reduce data overfitting.

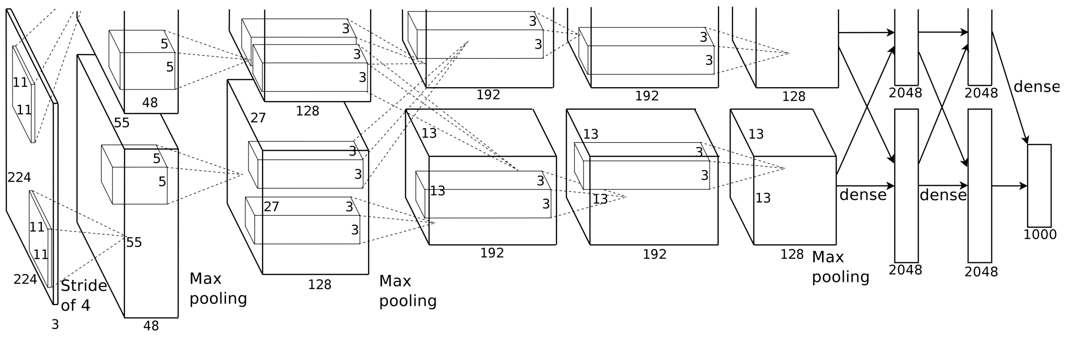


Figure 3.3: AlexNet architecture proposed by Krizhevsky *et al.* (2012) [3]. Figure originally found in [3].

3.1.3 VGG

Starting to increase the architectures' depth, the work of Simonyan *et al.* (2014) [17] explores the addition of convolutional layers with small convolutional filters. Of all the investigated networks, two had the top results: VGG-16 and VGG-19.

Both have an input size of $224 \times 224 \times 3$ but differ in the number of convolutional layers. While VGG-16 has 13 convolutional layers, VGG-19 has 16. Compared to AlexNet, VGG uses tiny receptive fields throughout the net, filtering the image with kernels of size $3 \times 3 \times 3$ with a distance between the receptive field centers of neighboring neurons in a kernel map of 1 pixel in the first layer. Also, there is not a direct alternation between convolutional and pooling layers.

3.1.4 GoogLenet

GoogLenet — or Inception — is a deep CNN model proposed by Szegedy *et al.* (2014) [4]. With 27 layers, Inception is a network with many parameters that try to work around the overfitting issue. Besides the traditional convolutional and pooling layers, there is the addition of a *Inception Module*. This module analyzes the correlation statistics of the last layer. It clusters them into groups of units with high correlation, forming the units of the next layer and are connecting them to the units in the previous layer. This module is built with a convolution layer of three different sizes of filters (1×1, 3×3 and 5×5) followed by a max-pooling layer. The filter outputs are concatenated and sent to the next layer. The naïvety of the described module leads to a need for dimensionality reduction to diminish the computational processing cost. To achieve this outcome, they used the idea of embeddings and added an extra convolutional layer with a filter size of 1×1 right before the 3×3 and 5×5 convolutions.

3.1.5 ResNet

ResNet [18] is also a deep architecture that relies in modules to prevent a learning degradation with the increase of depth. Instead of learning a direct mapping of $y = H(x)$ with a few stacked non-linear layers ($H(x)$), a residual mapping is proposed, where $F(x) := H(x) - x$, which can be reframed into $H(x) = F(x) + x$, where $F(x)$ and x represents the stacked non-linear layers and the identity function respectively.

According to He *et al.* (2015) [18], it is easier to optimize the residual mapping than the unreferenced mapping. Likewise, the formulation of $H(x) = F(x) + x$ can be realized by feedforward neural networks with shortcut connections. In the *ResNet Module*, these shortcut connections perform identity mapping, and their outputs are added to the outputs of the stacked layers.

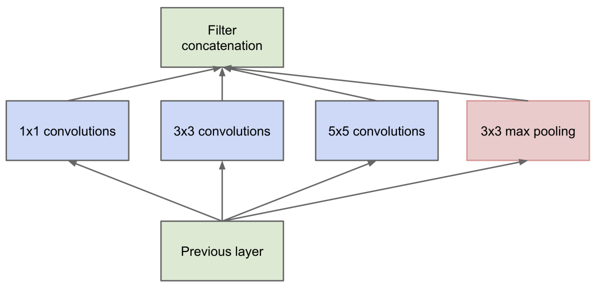


Figure 3.4: Naïve module.

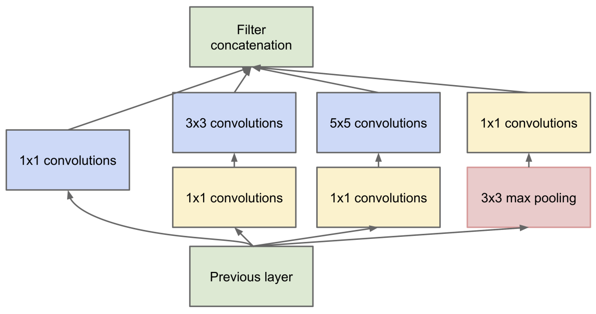


Figure 3.5: Module with dimensionality reduction.

Figure 3.6: Inception modules. Figure available in Szegedy *et al.* (2014) [4]. © 2014 IEEE.

3.1.6 Xception

Xception [53] network is an interpretation of Inception modules [4] with linear stacks of depthwise separable convolution layers with residual connections. The essential theory of Inception is that cross-channel and spatial correlations are reasonably decoupled. Xception's proposal is based on the hypothesis that the correlations above can be fully decoupled, modeling a network using 36 convolutional layers, structured into 14 modules with linear residual connections, leaving out the first and last modules.

3.1.7 DenseNet

In 2018, DenseNets or Densely Connected Convolutional Networks [54] were introduced. Analyzing that deeper CNNs were more accurate, Huang *et al.* (2018) presented an architecture that connects each layer to every other layer in a feed-forward way. The maximum information flow between layers in the network is guaranteed, promising to attenuate the vanishing-gradient issue and enhance feature propagation while using lesser parameters. In this model, the feature maps of all earlier layers are used as inputs for each layer. Their feature maps are used as inputs into all following layers.

Table 3.1: Literature review on CNNs, from oldest to newest references.

Year	Author	Description	Architecture	Dataset	Metric
1980	Kunihiko Fukushima [40]	Introduces CNNs	Neocognitron	—	—
1989	Alex Waibel *et al.* [42]	A primitive 1D ConvNet called a time-delay neural network was used for phoneme recognition	TDNN	Common Japanese words	acc: 98.50%
1990	Yann LeCun *et al.* [43]	The first paper using backpropagation on CNNs for the task of classifying handwritten digits	LeNet-1	USPS zip codes	acc: 96.60%
1990	Léon Bottou *et al.* [55]	A French digit recognition task with time-delay neural networks	TDNN	LIMSI Speech Database	acc: 95.00%
1997	Steve Lawrence *et al.* [56]	Presents a hybrid neural network, combining self-organizing maps (SOM) and CNNs, in order to recognize faces	SOM + CNN	ORL Database	acc: 96.20%
1998	Yann LeCun *et al.* [2]	This paper reviews various methods applied to handwritten character recognition and compares them, showing that CNNs outperforms all other techniques	LeNet-5	MNIST	acc: 99.05%
2012	Alex Krizhevsky *et al.* [3]	Use deep CNNs to almost halve the error rate in ImageNet challenge	AlexNet	ImageNet	acc: 84.70%
2013	Shuiwang Ji *et al.* [57]	Implement a 3D CNN model for action recognition, extracting features from both spatial and temporal dimensions by performing 3D convolutions	3D CNN	TRECVID 2008	auc: 0.0129%
2014	Nal Kalchbrenner *et al.* [58]	Describe Dynamic Convolutional Neural Network (DCNN) for the semantic modeling of sentences	DCNN	Stanford Sentiment Treebank	acc: 86.00%
2014	Yoon Kim [59]	Shows that a simple CNN with hyperparameter tuning and static vectors achieve good results on multiple benchmarks, especially for sentence-level classification tasks	Word vectors + CNN	Stanford Sentiment Treebank	acc: 48.00%
2014	Ross Girshick *et al.* [44]	The first paper to show that CNNs can lead to higher object detection performance as compared to systems based on simpler HOG-like features	R-CNN	PASCAL VOC 2010	mAP: 53.70%
2014	Karen Simonyan *et al.* [17]	Investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting	VGG	ImageNet	acc: 93.20%
2015	Jeffrey Donahue *et al.* [60]	Propose a novel recurrent convolutional architecture for large-scale visual learning	LRCN	UCF101	acc: 77.46%
2015	Christian Szegedy *et al.* [61]	A deep CNN with architectural decisions based on the Hebbian principle and the intuition of multi-scale processing, using the computing resources inside the network, was proposed. The depth and width of the network were increased while keeping the computational budget constant	GoogLenet	ImageNet	acc: 93.33%
2015	Ross Girshick [62]	Proposes a new training algorithm that fixes the disadvantages of R-CNN and improves its speed and accuracy	Fast R-CNN	PASCAL VOC 2010	mAP: 68.80%
2015	Shaoqing Ren *et al.* [37]	Introduce Region Proposal Networks (RPNs) that share convolutional layers with state-of-the-art object detection networks as Fast R-CNN	Faster R-CNN	PASCAL VOC 2012	mAP: 70.40%
2015	Kaiming He *et al.* [18]	A deep residual learning framework is proposed, reformulating the layers as learning residual functions concerning the layer inputs, instead of learning unreferenced functions	ResNet	ImageNet	acc: 94.29%

Year	Author	Description	Architecture	Dataset	Metric
2015	Du Tran *et al.* [63]	Propose an approach for spatiotemporal feature learning using 3D CNNs	3D CNN	UCF101	acc: 52.80%
2016	Leon Gatys *et al.* [64]	Use image representations derived from CNNs optimized for object recognition to render the semantic content of an image in different styles	VGG	—	—
2016	Mohammad Rastegari *et al.* [65]	Propose an architecture that uses bitwise operations to approximate convolutions, enabling the possibility of running state of the art deep neural network on CPU in real-time	XNOR-net	ImageNet	acc: 90.00%
2017	Kaiming He *et al.* [66]	Propose a general framework for object instance segmentation, efficiently detecting objects in an image while simultaneously generating a segmentation mask for each instance	Mask R-CNN	COCO	AP: 53.50%
2017	Andrew Howard *et al.* [67]	Present efficient models for mobile and embedded vision applications	MobileNets	ImageNet	acc: 70.60%
2017	François Chollet [53]	Shows and interpretation of the GoogLenet architecture (Inception), where Inception modules have been replaced with depthwise separable convolutions	Xception	ImageNet	acc: 94.50%
2018	Gao Huang *et al.* [54]	Introduce Dense CNNs, showing that CNNs can be deeper, more accurate and efficient to train if they contain shorter connections between layers close to the input and those close to the output	DenseNet	ImageNet	acc: 94.71%

3.2 Multistream Convolutional Neural Networks

Karpathy *et al.* (2014) [6] proposed an empirical evaluation of CNNs on video classification. Noticing that the standard video classification approaches [68, 69, 70, 71, 72, 73, 74] consisted in involving three main stages (visual features extraction, feature combination, and then classification) that CNNs were achieving state-of-the-art results on image recognition, segmentation, detection, and retrieval [3, 75, 76, 77, 78, 44], a new technique that included local motion information in the video as connectivity to a CNN architecture was suggested.

Considering that there is the presence of the time domain in this type of application, a multiresolution architecture was studied to extend the connectivity of a CNN and use the additional information to improve overall performance. Therefore, multiple CNN architectures with different approaches to combining information are evaluated. These architectures are detailed in Section 3.2.1.

As stated by Ross Girshick *et al.* (2014) [44]: *Features matter*. Bringing the multistream concept from video classification to image classification, we can input almost any extra information in the other streams, such as a segmented version of the first input or a crop of a region of interest. The chart presented in Figure 3.7 shows the growth of the *Multistream CNN* topic from 2009 to 2020. Our consulted sources were three of the most important digital libraries, cited in Section 2.2. Also, Table 3.2 shows a timeline of the theme.

A multistream approach can speed up the runtime of the network while increasing the

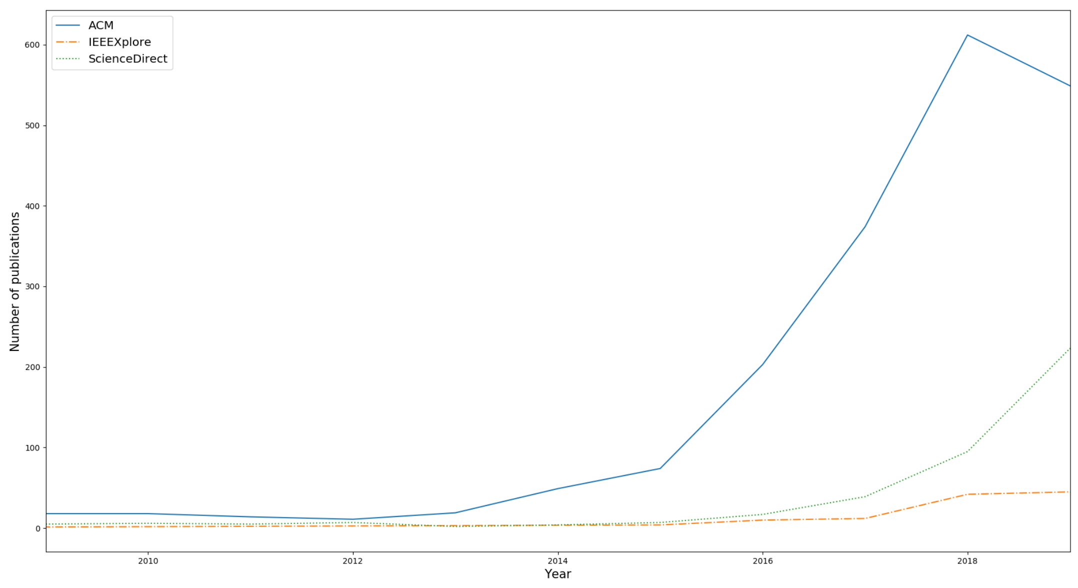


Figure 3.7: Occurrences of the term 'M-CNN' in 3 different publishers.

performance, as it can be seen in [11, 12], due to the additional relevant information injected at the beginning of the process; and the most important: it adds more features to be learned.

In Simonyan *et al.* (2014) [7], there is the proposition of a two-stream CNN architecture for action recognition, one spatial and one temporal stream. Also, they demonstrate that the use of this technique gives good results, even with limited training data. An approach similar to Karpathy *et al.* (2014) [6] is adopted, the Late Fusion (detailed in Sub subsection 3.2.1.1). The temporal stream transmits the camera movement while the spatial flow carries information about scenes and objects. Although they use Late Fusion, two fusion methods are considered: averaging and training a multi-class linear SVM on stacked $L2$-normalized softmax scores as features [7].

Still on the video classification problem, Wu *et al.* (2015) [79] propose a framework that integrates four different streams: spatial, short-term motion, long-term temporal, and auditory clues in videos. They also concluded that the extra information could boost the performance. Again, a Late Fusion model is implemented with a fusion method that learns weights adaptively for each class.

Inspired by the work of Simonyan *et al.* (2014) [7], Feichtenhofer *et al.* (2016) [28] propose a fusion evaluation of CNNs, to best take advantage of the additional information that this kind of network disposes of. Using the two-stream architecture previously proposed by Simonyan *et al.* (2014) [7], the authors consider different types of fusion, for spatial — sum, maximum, concatenation, convolution, and bilinear — and temporal — 3D pooling and 3D convolution + pooling — features.

In Gammulle *et al.* (2017) [27] they focused on learning salient spatial features via a CNN and then mapped their temporal relationship with a Long Short-Term Memory (LSTM) network. There are two different models that fuse in different ways, being a simple merge and the use of another LSTM that generates one single output. According to Gammulle

Table 3.2: Literature review on M-CNNs, from oldest to newest references.

Year	Author	Description	Architecture	Dataset	Metric
2014	Andrej Karpathy *et al.* [6]	Introduces M-CNNs. A multiresolution network that has an AlexNet-like [3] CNN attached to its firsts layers	M-CNN	UCF-101	mAP: 66.00%
2014	Karen Simonyan *et al.* [7]	Propose a two-stream CNN architecture which incorporates spatial and temporal networks for action recognition	M-CNN	UCF-101	acc: 88.00%
2015	Zuxuan Wu *et al.* [79]	Train three CNNs to model spatial, short-term motion and audio clues. Then, use LSTM networks to explore long-term temporal dynamics. With the outputs of the individual streams, they propose a fusion method to generate the final predictions	M-CNN	UCF-101	acc: 92.20%
2015	Limin Wang *et al.* [80]	Design good practices for the training of very deep two-stream CNNs in the video action recognition domain	M-CNN	—	—
2016	Jingxiang Yang *et al.* [81]	Use a two-channel CNN to learn jointly spectral-spatial feature from hyperspectral image	M-CNN	Salinas Valley + Indian pines	acc: 95.58%
2016	Xiaojiang Peng *et al.* [82]	Propose a two-stream R-CNN for video action detection, showing that a motion region proposal network generates high-quality proposals and that stacking optical flow over several frames significantly improves frame-level action detection	M-RCNN	UCF-Sports	acc: 95.74%
2016	Ali Diba *et al.* [83]	Explore motion representation with 3D-CNNs, learning distinctive models to combine deep motion features into appearance model via learning optical flow features inside the network	M-3DCNN	UCF-101	acc: 90.20%
2016	Bharat Singh *et al.* [84]	After locating bounding boxes around a person, they train two additional streams on motion and appearance cropped to the tracked bounding box, along with full-frame streams	MRNN	Shopping Dataset	acc: 80.31%
2016	Christoph Feichtenhofer *et al.* [28]	Study a number of ways of fusing CNNs both spatially and temporally in order to best take advantage of the spatio-temporal information	M-CNN	UCF-101	acc: 90.62%
2016	Velickovic *et al.* [8]	Propose a network with cross-connections inserted after pooling operations and full weight sharing in the fully connected layers	X-CNN	CIFAR-10	acc: 86.14%
2017	Dahjung Chung *et al.* [85]	Propose a two stream architecture where each stream is a siamese network to the person re-identification task in video surveillance systems	M-CNN	iLiDS-VID	Matching acc: 97.00%
2017	Harshala Gammulle *et al.* [27]	Evaluate a deep fusion framework that exploits spatial features from CNNs with temporal features from LSTM models	Two-stream LSTM	UCF-11	acc: 94.60%

Year	Author	Description	Architecture	Dataset	Metric
2017	T. Akilan *et al.* [86]	Explore late fusion upon different multi-deep CNNs used as feature extractors	M-CNN	Caltech101	acc: 95.54%
2018	Yunlong Yu *et al.* [87]	Use M-CNNs to classify aerial scenes, with two pretrained CNNs as feature detectors	M-CNN	UC-Merced	acc: 97.12%
2018	Miao Ma *et al.* [5]	Propose a method that reduces feature dimension while also effectively combining appearance and motion information in a unified framework	M-CNN	sub-JHMDB	acc: 76.90%
2018	Qi Xuan *et al.* [29]	Use M-CNNs to classify pearls with multiview images	MS-CNN	Pearls labelled by experts	acc: 92.14%
2018	Darwin Concha *et al.* [30]	Use M-CNNs for action recognition in video sequences using spatial and optical flow information	M-CNN	UCF-101	acc: 94.30%
2018	Chenjie Ge *et al.* [32]	Use M-CNNs to extract and fuse the features from multiple sensors for glioma tumor grading	M-CNN	MICCAI BraTS 2017 competition	acc: 90.87%
2018	Petar Veličković *et al.* [88]	Analyze multi-modal time-series data with a cross-fused three-layer LSTM	X-LSTM	Nokia Digital Health - Withings	acc: 80.30%
2019	Zhigang Tu *et al.* [31]	Use M-CNNs to recognize action in videos by using semantics-derived multiple modalities in both spatial and temporal domains	M-CNN	UCF-101	acc: 94.80%
2019	Mostafa Amin-Naji *et al.* [89]	Use M-CNN to create multiple focuses on the dataset, using a concatenation as a fusion method	M-CNN	COCO	acc: 99.78%
2019	Juan-Manuel Pérez-Rúa *et al.* [90]	Propose a technique to search an optimal architecture for a fiven dataset, looking for different possible fusions	MFAS	NTU RGB+D	acc: 93.46%
2019	Ana Paula Almeida and Flavio Vidal [91]	Present a CNN-based model with a cross-fusion method for multiple streams applied for image classification. This architecture is focused on the streams' fusion stages	L-CNN	CIFAR-10	acc: 62.57%
2020	Cătălina Cangea *et al.*	Join two architectures with multimodal cross-connections, allowing a dataflow between several feature extractors	XFlow	AVletters	acc: 89.60%
2020	Hamid Reza Vaezi Joze *et al.* [9]	Create a fusion module that uses squeeze and excitation operations to recalibrate features on the streams	MTMM	NTU RGB+D	acc: 90.11%

et al. (2017) [27], the motivation behind having multiple layers of LSTMs is to capture information hierarchically. Each stream can communicate with each other and improve the backpropagation process.

More recently, a six-stream network was proposed by Ma *et al.* (2018) [5]. Based on an estimation of human pose and human parts positions in video sequences, different patches of crops serve as input to the network, Figure 3.8. There is no elaborated fusion technique in this work, just a concatenation of the outputs before the first fully connected layer.

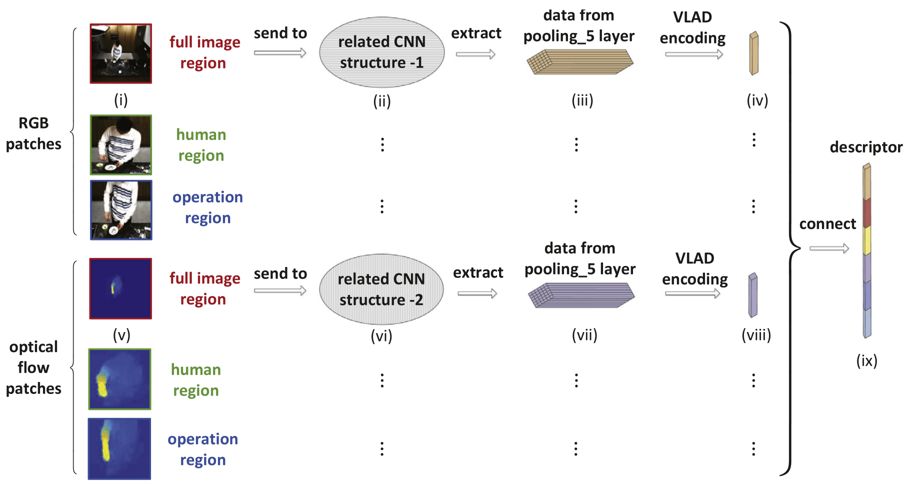


Figure 3.8: Six-stream CNN. Adapted from Ma *et al.* (2018) [5].

It is still unusual to work with multiple streams in an image classification context. Aerial images, for example, have not only space and texture features but also contain a large number of scene semantic information [87]. Thus, a good feature representation is needed for positive classification results. Yu *et al.* (2018) [87] proposed a two-stream CNN based on two pre-trained CNNs as feature extractors to learn deep features from the original aerial image and the processed aerial image through saliency detection, respectively. Right before an extreme learning machine classifier, the streams are fused in two distinct strategies – concatenation and sum.

3.2.1 M-CNN Models and Architectures

M-CNNs are directly derived from traditional CNN architectures. This Section explores some successful models with more than one input and presents our model proposal.

3.2.1.1 Karpathy's two-stream CNN

According to Karpathy *et al.* (2014) [6], the Single Frame architecture is a traditional single-stream CNN, which can be any known architecture. Their model uses a single-stream AlexNet [3] as a baseline. Considering that videos vary in temporal extent and each clip has several frames, extending the network's connectivity in the time dimension is possible learning spatio-temporal features. Figure 3.9 is illustrates their proposed connectivities.

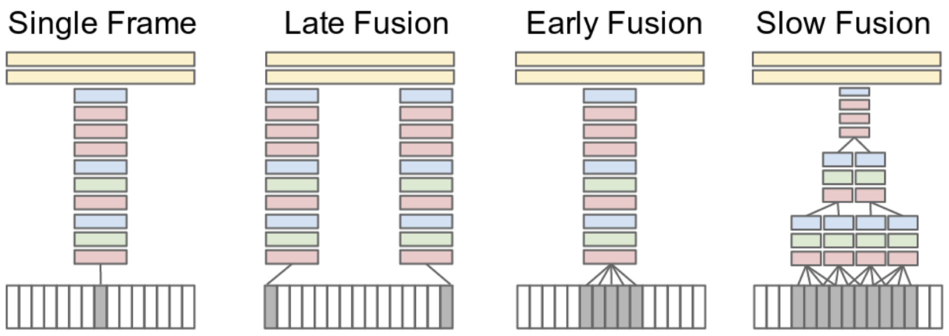


Figure 3.9: Evaluation of multiple CNN architectures proposed by Karpathy *et al.* (2014) [6]. Red, green and blue boxes represent convolutional, normalization and pooling layers respectively. Figure taken from [6]. © 2014 IEEE.

Although it looks similar to the Single Frame model, the Early Fusion model combines information across an entire time window immediately on the pixel level by modifying the filters on the first convolutional layer in the first shown model. This connectivity allows the network to detect local motion direction and speed precisely.

The Late Fusion architecture places two (or more) single-stream networks with shared parameters and then merges the two streams in the first fully connected layer. Therefore, the first fully connected layer can compute global features by comparing the outputs of both towers.

The Slow Fusion model is a balanced mix between the two multi-stream approaches that slowly fuses information throughout the network. Higher layers get access to progressively more global information in both dimensions.

Lower resolution images were used in a multiresolution architecture to speed up the models' runtime performance with accuracy maintenance. Two separate streams served as an input to the M-CNN. The context stream received the downsampled frames at half the original spatial resolution. The fovea stream received the central region at the original resolution, taking advantage of a usual camera bias.

3.2.1.2 Simonyan's two-stream CNN

Simonyan *et al.* (2014) proposed a two-stream architecture to receive spatial and temporal components of a video. Each stream is composed of a deep CNN, as detailed in Figure 3.10. All hidden weight layers use the rectification (ReLU) activation function, and max pooling is performed over 3×3 spatial windows with stride 2. The fully convolutional layer is combined by using the previously cited Late Fusion.

In this work, two fusion methods are considered: averaging and training a multi-class linear SVM on stacked $L2$-normalized softmax scores as features.

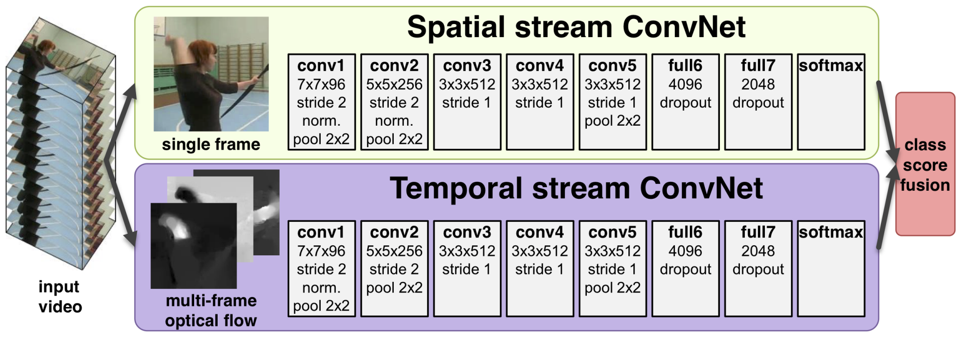


Figure 3.10: Two-stream architecture proposed by Simonyan *et al.* (2014) [7]. Figure taken from [7].

3.2.1.3 X-CNNs

As previously shown, multiple stream networks have been developed, applied, and used in many situations and applications nowadays [27, 29, 30, 31, 32, 33]. Nevertheless, signal treating in between multistream layers strategies are lacking in the literature. Velickovic *et al.* (2016) [8] present a cross-modal architecture for image classification, the X-CNN.

Designed to deal with sparse data sets, X-CNNs are a typically image-based approach that allows weight-sharing and information exchange between hidden layers of a network by using cross-connections inserted after each pooling layer, presented in Figure 3.11. Inspired by the biological cross-connections between various sensory networks, this strategy attempts to improve CNNs predictions without requiring large input data dimensionality.

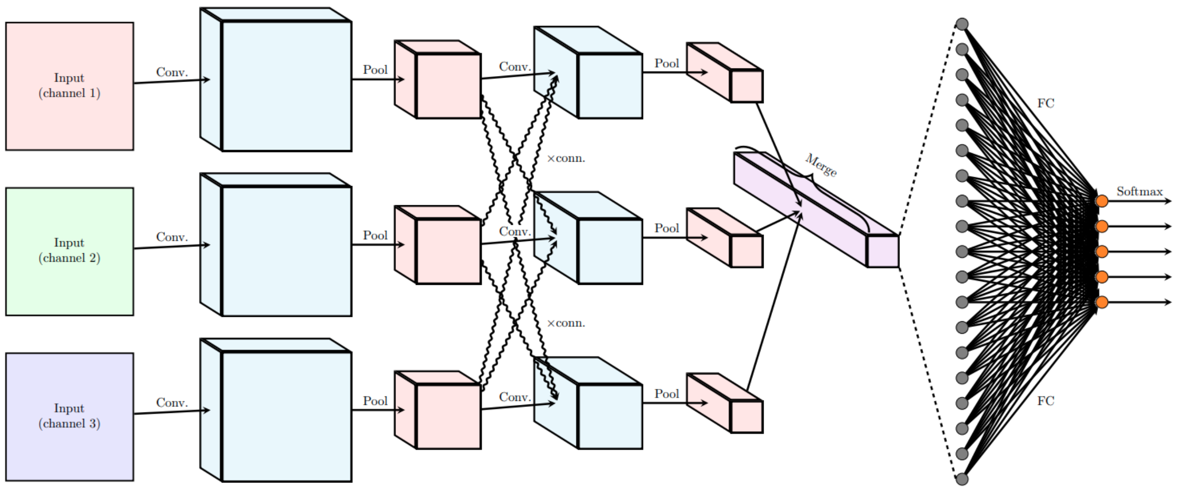


Figure 3.11: X-CNN architecture. Figure originally found in Velickovic *et al.* (2016) [8].

3.2.1.4 M-CNNs as ensembles

Akilan *et al.* (2017) [86] explore late fusion upon different multi-deep CNNs used as feature extractors. Their approach uses a four rule feature fusion – product, sum, average and maximum – to merge different CNN features. Based on previous works that ensembled distinct classifiers, such as K-Nearest Neighbors (KNN) and CNNs, they demonstrate that even simple fusion processes can improve classification results.

Still following the line of multiple CNN architectures ensemble, Amin-Naji *et al.* (2019) [89] also use several networks simultaneously trained on a dataset to solve an issue. This time, however, the proposed methodology also intends to create multiple focuses on the dataset, improving the overall accuracy. For fusing the networks, a concatenation strategy is used.

3.2.1.5 Multimodal M-CNNs

It is typical to observe multiple streams when treating videos with spatial and temporal portions or when using different data modalities. In Veličković *et al.* (2018) [88], multimodal time-series data is analyzed using an X-LSTM technique. Each input passes through a separate three-layer LSTM stream, allowing a piece of information to flow using cross-connections between the streams in the second layer, where features from a stream are passed and concatenated with features from another stream. In Pérez-Rúa *et al.* (2019) [90], they propose a search space that covers numerous possible fusion architectures given the nature of the multimodal dataset. The outputs of layers that perform a function, such as convolutions or poolings, are then eligible to be chosen for fusion in this approach. When a fusion point is selected, a concatenation operation is performed. The XFlow network [92] also brings an ensemble of different architectures, but with cross-connections as fusing strategies.

In 2020, Joze *et al.* [9] proposed a multimodal transfer module for CNN fusions – MTMM –. The MTMM can be added at different levels of the model, and one main advantage is that the input tensors do not have the same spatial dimensions, as it performs squeeze and excitation operations. Also, each stream can be initialized with existing pre-trained weights since minimum chances are made in the main structure. Although the module is not mainly used for image classification, it can be adapted to this task. An illustration of the module can be found in Figure 3.12, where A and B are layer features.

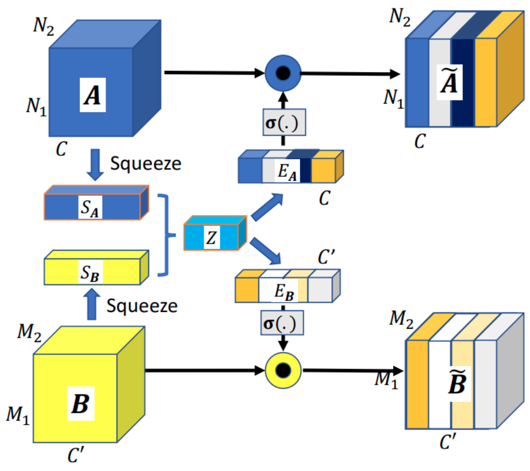


Figure 3.12: MTMM module. Figure originally found in Joze *et al.* (2020) [9].

After presenting in this Chapter a list of main topics and research related to artificial neural networks, focused on the convolutional and multistream architectures, in the next chapter, we will introduce our proposed methodology to prove the Hypothesis H1.

Chapter 4

Methodology

As stated in Section 1, the main goal of this work is to reach a new topology that involves all the knowledge gathered during this time. To achieve the expected result, some steps were taken and are shown in Figure 4.1.

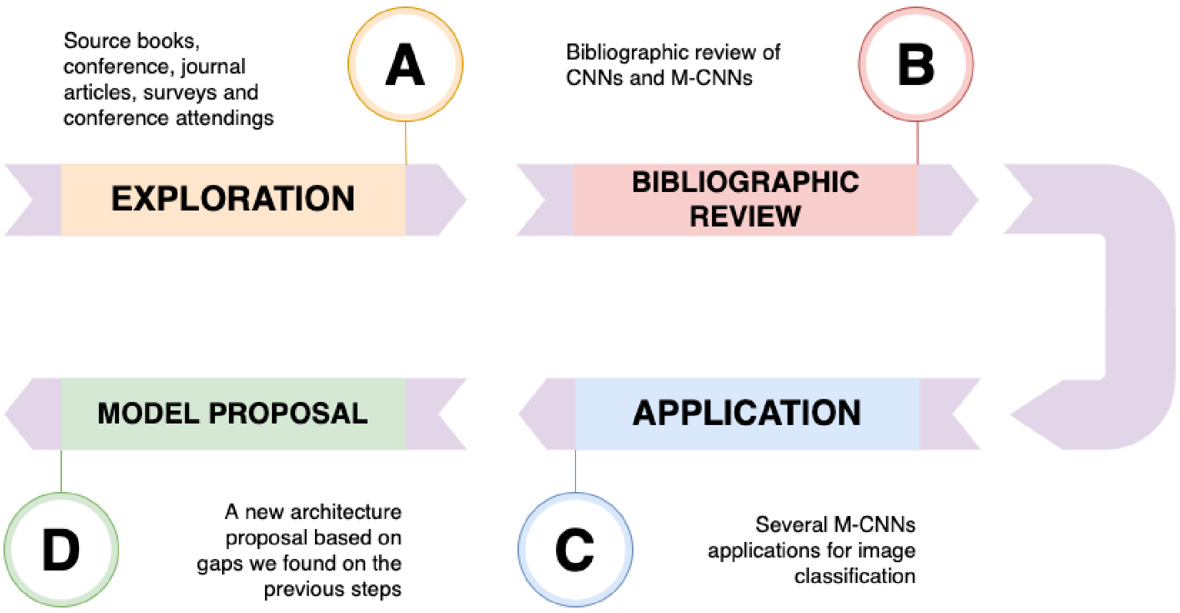

Source books, conference, journal articles, surveys and conference attendings
A
EXPLORATION
Bibliographic review of CNNs and M-CNNs
B
BIBLIOGRAPHIC REVIEW
MODEL PROPOSAL
D
A new architecture proposal based on gaps we found on the previous steps
APPLICATION
C
Several M-CNNs applications for image classification

Figure 4.1: Methodology workflow.

The initial step of this work was an exploratory methodology. As a primary reference source, we used books, conferences, and journal articles [1, 2, 4, 3, 6, 15, 93]. As secondary, survey articles and participation in conferences in the area of artificial intelligence and computer vision were used [11, 12, 16]. This phase was fundamental to understand what difficulties and benefits the area is going through. After the bibliographical survey, it was possible to verify that the scope of CNNs has been gaining more and more space among the scientists and that special attention to M-CNNs has been given since it was an approach that could handle more features at once.

Besides being exploratory, the proposed methodology is also sequential. The following steps were taken: A bibliographic review of CNNs and M-CNNs was performed to discover the main gaps in the literature and define the line of work to be filled by this dissertation. It was also important to determine the intermediate steps to accomplish the primary goal. Also, we realized a study of the main CNN (*e.g.* GoogLenet [4], Alexnet [3]) and M-CNN (*e.g.* Multistream Networks [6], Two-Stream Network [7]) architectures.

In addition to the previous steps, the implementation of the networks above for different applications, such as image and document classification was made. This step was fundamental to understanding how the main frameworks and APIs work (specifically Keras [94] and Tensorflow [95]). Moreover, the first contributions were produced — mainly for M-CNNS.

Starting with video classification, we learned the most basic techniques of M-CNNs. How to use multiple streams, handle these various streams, combine them, and compare their result with a single stream network.

Understanding the impact of an additional stream on a neural network, we extrapolate the problem to a previously unexplored type of application: image classification. Using a dataset of plant diseases, we transformed the original dataset into versions that empirically could contribute to the feature extraction modules of a deep neural net. Again, we got positive results with the multiple streams strategy.

Having confirmed that the technique also works for image classification, we wondered if the fusion of two different modalities could also present a performance gain. Here we worked with image and text, two forms of input with additional information about the same problem. As we entered this field, we saw that some efforts towards multimodality were already being explored. Still, we decided to focus on our previously learned fusion techniques and models and compare them with the state-of-the-art.

Later on, we gathered our efforts towards a different area of interest and tested our proposed method on object detection problems. We still have a lot to adjust, but preliminary results are promising.

It is essential to notice that we chose to work on non-standard problems: M-CNNs were, at first, commonly used for video classification, where we have spatial and temporal features to analyze, and we decided to apply most of our adaptations to image classification issues. This choice was conscious as we wanted to check if the growth of this kind of network could improve the accuracy of other modalities, always with a concern of providing a good second stream for the network.

At the same time that multimodality was explored, another aspect was also being studied within image classification. While we constantly noticed the gain in accuracy, sometimes this gain was not that significant or required a computational power that was too high for us to be able to start the experiment with multiple streams. Thus, one way to continue applying this technique was to examine fusion points that would increase accuracy with less training time and would not increase the network parameters to make it impossible to run tests.

All applications are further detailed in Chapter 5.

4.1 Topology proposal

Creating M-CNN versions of classic architectures was a great learning experience in the use of programming libraries and tools and in understanding the main issues of using multiple streams. We started to inquire more about what would be a perfect complement to standard RGB stream and at what moment the fusion would work better in that gigantic new structure. These questions became the next point of reflection in this work.

After the exploration, bibliographic surveys, and practical implementations, the definition of the problem took shape, and we could identify a central gap in the literature: where to fuse M-CNN models. Also, we had the restriction of not having infinite computational power to run experiments.

This strategy should be usable in lots of different networks without growing the parameters excessively.

Considering the restrictions mentioned above and that the ReLU activation on convolutions is widespread and its implementation is simple, a fusion strategy for any model containing ReLU's activation was proposed, developed, and tested. To do so, we searched for different application scopes: traditional datasets, such as CIFAR-10 [10], were used to validate the novel method. Furthermore, we also analyzed our proposition with a complex image classification dataset called NORB [96]. Subsequently, we expanded the implementation to object detection using the BDD100K dataset [97]. More details of the implemented strategy and its results can be found in Chapter 6.

4.1.1 Model recycling

While designing the new module, we observed that our strategy was able to boost results from outdated architectures, making them competitive again.

As a consequence of this behavior, we were able to compare the implementation of our technique both in more traditional and no longer used models and in more recent models. And the results showed that older and simpler models could be used to solve current problems with compatible or close to state-of-the-art performance.

One of the most common complaints currently in deep neural networks is the indiscriminate use of increasingly complex and, thus, irreproducible models by institutions or small companies. Very complex models require more powerful hardware, which, in turn, uses much more energy and natural resources.

Nonetheless, our strategy can be applied to revive simple structures with the advantage of great assertiveness using less hardware and therefore fewer assets. All these comparisons and results can be found in detail in Chapter 6.

4.2 Datasets

To achieve the results presented in Chapters 5 and 6, we describe here the used data for all applications.

4.2.1 The comma.ai driving dataset

The *comma.ai* organization dataset is composed of 11 videos with a 320×160 pixels resolution, variable duration and recorded in a 20Hz frequency using a dash camera installed inside the vehicle, capturing a total of 522,434 frames of its frontal vision while driving during day and night on highways, corresponding to approximately 7 hours of recording time. An example frame can be found in Figure 4.2.

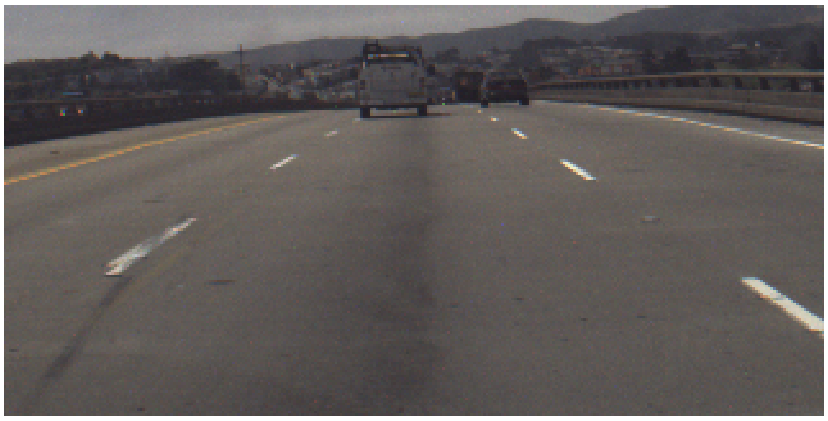


Figure 4.2: A frame found in comma.ai dataset.

For the purposes of the *Autonomous vehicle steering wheel estimation* work, the only information that was taken into account was the video frames in RGB format and the steering angle values. The steering angle is given in degrees and corresponds to the rotation angle of the vehicle's steering wheel. Every angle value present in the dataset is in the interval $[-502.3, 512.6]$.

Thus, each training session was repeated 3 times and each time the dataset was randomly partitioned in training/validation/test subsets, following a 70/15/15 proportion schema.

4.2.2 PlantVillage

The PlantVillage dataset, provided by [98], containing 54,306 images of plant leaves and 38 different classes with 14 plant species, each class corresponding to a different crop disease. Every class has three different versions: original colored image, grayscaled image and segmented image. Figure 4.3 shows samples from the data.

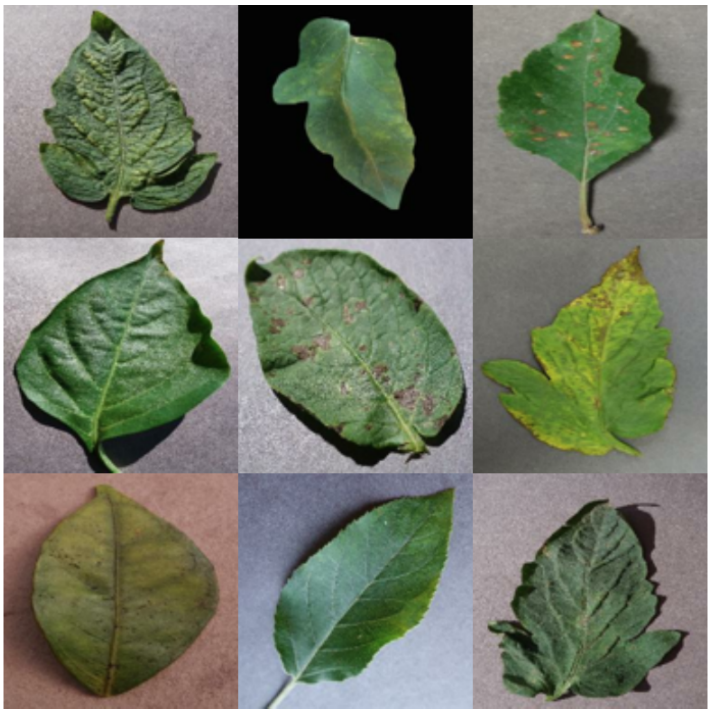


Figure 4.3: A frame found in comma.ai dataset.

The set proportion was 80% of the whole dataset used for training, and 20% for testing, as suggested by Mohanty *et al.* (2016) [99].

4.2.3 VICTOR

The VICTOR dataset [100] is composed of five main types of legal documents that make up the cases that are dealt by the Supreme Federal Tribunal of Brazil.

Containing 6,814 manually annotated documents with over 1,000,000 pages, it is generated a stratified split for each document class, maintaining the proportions of class samples in each subset, resulting in the proportions of 70% for the training set, 20% for validation and 10% for the test set.A sample of these documents can be found in Figure 4.4. They are intentionally blurred so that critical information is hidden.

4.2.3.1 Small VICTOR

Being a subset of VICTOR, the small VICTOR dataset limits the number of suits for each theme to 100 samples in each set, which contains 6,510 Extraordinary Appeals, 94,267 documents and 339,478 pages.

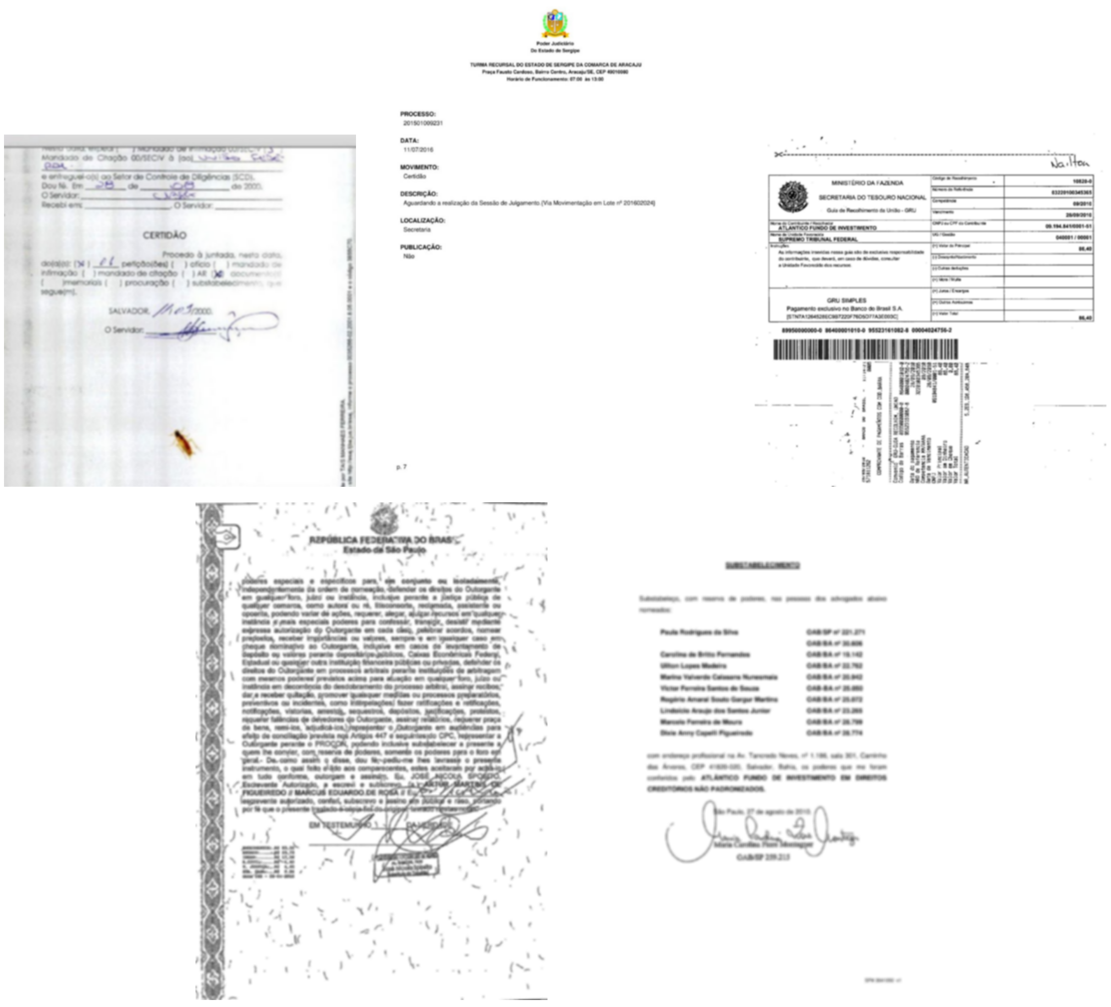


Figure 4.4: Blurred document samples from the VICTOR data set.

4.2.4 CIFAR-10

Divided into five training batches and one test randomly generated batch, each with 10000 images, CIFAR-10 [10] consists of 60000 32x32 color images in 10 classes, with 6000 images per class. Figure 4.5 shows ten samples of each class.

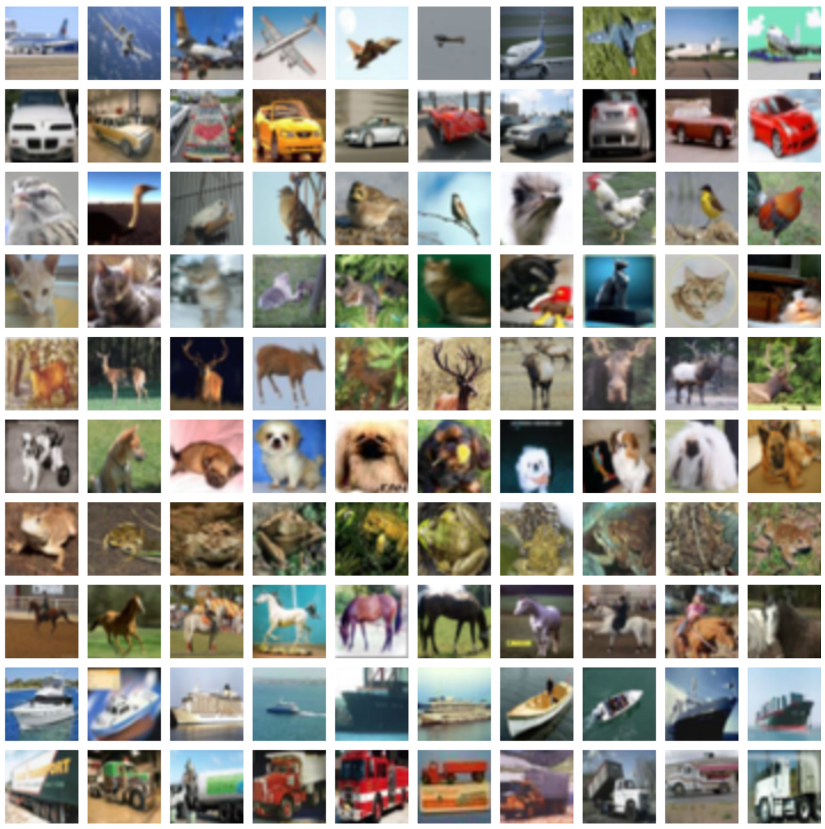


Figure 4.5: CIFAR-10 samples. Original figure found in Krizhevsky (2019) [10].

4.2.4.1 CIFAR-50 and CIFAR-100

Instead of 10 classes, as in CIFAR-10, the CIFAR-100 has 100 classes with 600 32x32 images in each class. Inside the one hundred classes, there are 20 superclasses that generalize certain labels, but these were not considered in this work.

To produce the CIFAR-50 dataset, we randomly chose fifty classes from the original CIFAR-100 dataset. The final samples consisted of 50 classes with 600 images each, 500 for training and 100 for testing.

4.2.5 NORB Object Recognition Dataset

The main goal of this dataset is to recognize 3D objects from shapes. It has 48600 pictures of 50 toys belonging to 5 general categories: airplanes, four-legged animals, trucks, human figures, and cars. The dataset provides a pair of images for every toy: the items were imaged by a pair of cameras under 9 elevations (30 to 70 degrees every 5 degrees), 6 lighting conditions, and 18 azimuths (0 to 340 every 20 degrees) [96].

Training and validation sets are equally divided, being 5 instances of each category (4, 6, 7, 8 and 9) selected to the former and the remaining instances for the latter.

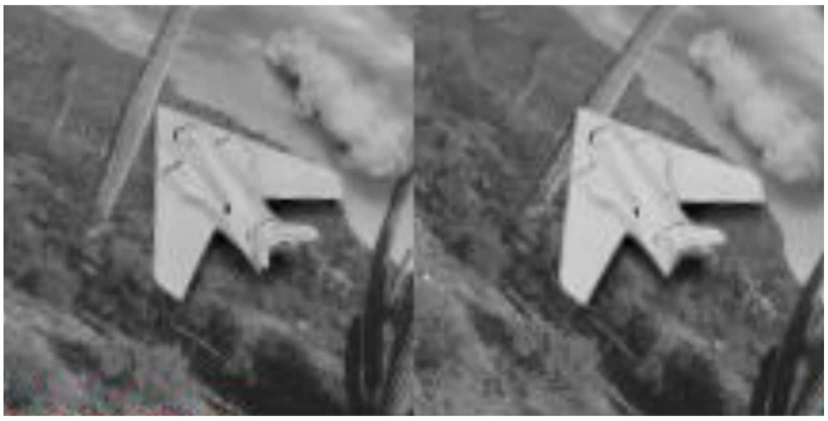


Figure 4.6: Pair of images found in NORB dataset.

4.2.6 BDD100K

BDD100K [97] is the largest open driving video dataset, containing $100,000$ high-resolution videos and bounding box annotations of 10 categories (road, sidewalk, building, wall, fence, pole, light, sign, vegetation, terrain, sky, person, rider, car, truck, bus, train, motorcycle, and bicycle) to evaluate image recognition algorithms on autonomous driving. The videos are split into $70,000$, $20,000$ and $10,000$, being, respectively, training, validation and test sets. Figure 4.7 shows one frame of a video containing almost all of the categories.

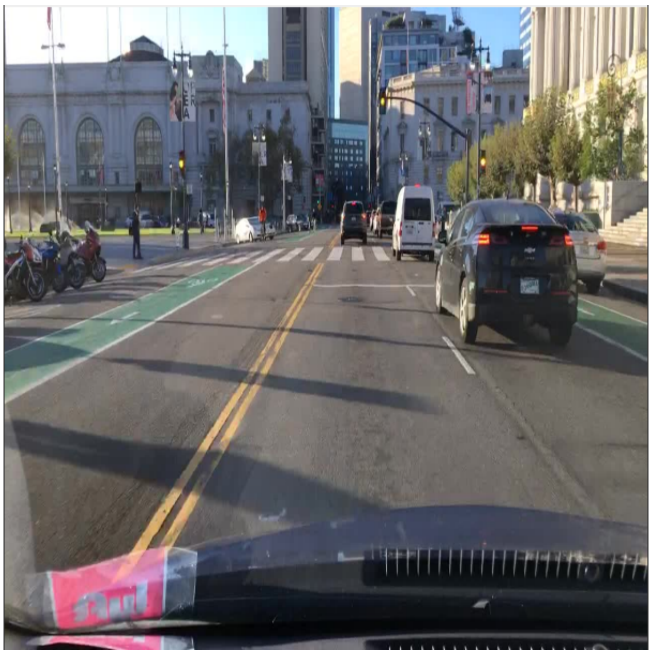


Figure 4.7: A frame of the BDD100K dataset.

4.3 Metrics

The evaluation metrics commonly used to check the quality of results obtained in a machine learning experiment are precision, recall, and F-measure [101].

Binary measures are defined to calculate these metrics, namely: the true positive (tp), which represents an item being correctly considered relevant; false positive (fp), when an item is erroneously computed as relevant; true negative (tn), representing an object correctly considered irrelevant and false negative (fn), where an item is erroneously classified as irrelevant.

Recall represents the proportion of positive cases that were correctly predicted as such, however, it is not usually used as an evaluation metric because it does not give any trace of the items that were not returned [101]. Equation 4.1 defines recall as:

$$recall = \frac{tp}{tp + fn}. \tag{4.1}$$

Precision denotes the proportion between predicted positive cases and those that are actually positive and is represented by the Equation 4.2.

$$precision = \frac{tp}{tp + fp}. \tag{4.2}$$

In addition to using precision, several authors [44, 62, 37] represent data analysis with a metric derived from precision, the mean average precision (mAP). This metric corresponds to the average sum of all precisions in a given data set and is described in Equation 4.3.

$$mAP = \frac{1}{ns}\sum_{i=1}^{ns} precision_i, \tag{4.3}$$

where ns is the number of samples.

A harmonic mean between these two measures generates the F-measure, presented in Equation 4.4. However, in the same way as the previous tests, the F-measure also does not consider the items negatives that were correctly classified as negative.

$$fmeasure = 2.\frac{precision.recall}{precision + recall}. \tag{4.4}$$

Accuracy has the full scope of information about the data, both positive, the successes, and negative, the errors, and is given by Equation 4.5, where acc is the accuracy.

$$acc = \frac{tp + tn}{tp + tn + fp + fn}. \tag{4.5}$$

The error measurement generally used to evaluate the prediction of numerical values is the root of the mean squared error (RMSE), which is described by Equation 4.6.

$$RMSE = \sqrt{\frac{1}{ns}\sum_{i=1}^{ns}(a_i - a'_i)^2}, \tag{4.6}$$

where ns corresponds to the number of predictions, g$_i$ equals to the ground truth of the i-th example and g'$_i$ is the prediction. To enable the comparison between different combinations of subsets, the normalized form of the RMSE (NRMSE) was chosen. It is evaluated in Equation 4.7:

$$NRMSE = \frac{RMSE}{g_{max} - g_{min}}, \tag{4.7}$$

where a_{min} and a_{max} correspond to the lowest and greatest values observed.

Chapter 5

Initial Contributions and filling gaps

Following the methodology presented in Chapter 4, and having completed stages A and B, as demonstrated in Chapters 2 and 3, respectively, this Chapter describes the initial results of the applications carried out during the study. The results presented here are part of the maturation process of the studies carried out previously, which allowed the advancement to the next stage in the proposed methodology proposing the new model.

The contributions presented in the following Sections are all focused on using M-CNNs, which allowed the production of expressive results that guided the work carried out, including publications in high-impact vehicles (conferences and international journals), describe in Chapter 7.

5.1 Autonomous vehicle steering wheel estimation

Nowadays, autonomous driving and navigation technology is an artificial intelligence application that has drawn significant attention with the popularity of intelligent vehicles. In these application areas, unsolved issues remain and have been significantly explored by the automotive and technological industries due to the potential impact that such innovation will bring in the near future [102, 103, 104].

Over the last decades, many works have approached this theme: In 1989, Pomerleau [102] described the construction of an autonomous vehicle based on artificial neural networks. The proposed network is responsible for providing guidance to the vehicle, and the architecture of the network consists of a classical artificial neural network (ANN) with a single intermediate layer, containing 29 neurons fully connected to the input units.

Many works involving ANN in autonomous vehicle applications can be found in the available literature. However, a lot has changed since the advent of new network architectures. With the rise of deep learning, CNNs have improved the image comprehension tasks by learning more discriminative features, allowing a proper development on several systems, including autonomous vehicles [105].

Using this motivation as a basis, we proposed in Ferreira *et al.* (2018) [11] a M-CNN capable of estimating the steering angle of an autonomous vehicle, having as only input images captured by a camera attached to the vehicle's frontal area.

Following a single stream CNN architecture, we proposed a multichannel model that processed the inputs in different channels until the last exponential linear unit (ELU) layer. In other words, the outputs of the last ELU layers from each channel are concatenated and passed as input to the first fully connected layer, as shown in Figure 5.1.

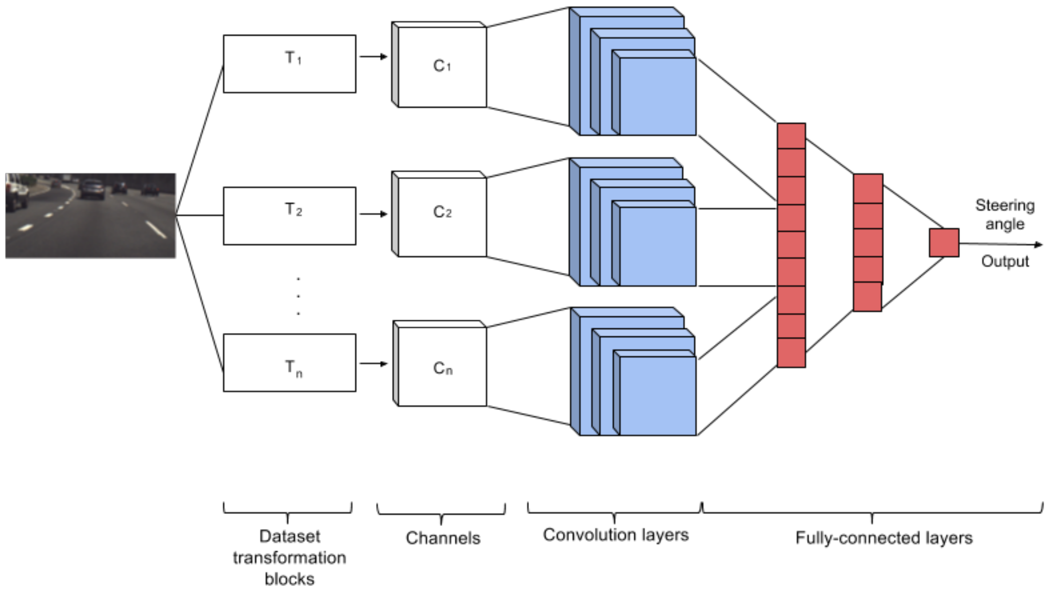


Figure 5.1: Proposed M-CNN architecture. Adapted from Ferreira *et al.* (2018) [11].

The original *comma.ai* dataset, described in Chapter 4, had to be adapted to generate different inputs to the M-CNN model. Thus, using Karpathy *et al.* (2014) [6] fovea input as inspiration, we created two more versions of the data besides the original video: a frame subsampled in 50% and central region of the image in the original scale.

With the new dataset versions, five model combinations were designed. **Model 1** was the original single-channel CNN. **Model 2** used the two-channel CNN architecture, receiving the original image and its frame subsampled in 50%. **Model 3** also used the two-channel CNN architecture, but it received the original image and its central region. **Model 4** uses the three-channel CNN architecture, aiming to observe whether the neural network can produce better results if the additional information is combined with the original frame. Another model, **Model 5**, containing only the generated versions of the dataset, was also evaluated.

Table 5.1: Average test errors in percentage (%) of all trained models. Table adapted from Ferreira *et al.* (2018) [11].

	Number of epochs		
	200	**350**	**500**
Model 1	5.75	4.72	4.80
Model 2	**4.09**	4.63	4.79
Model 3	4.85	**3.80**	**4.76**
Model 4	4.75	4.69	5.11
Model 5	4.89	5.12	5.27

As seen in Table 5.1, it was possible to observe that **Model 3** trained with 350 epochs obtained a lower NRMSE when compared to the others, including the single channel as a reference model. It can also be seen that M-CNNs provided improvements in their use in autonomous vehicle applications, with an approximate gain of 7% compared to the reference model. Furthermore, **Model 5** presented similar results to the others, showing that it was capable of maintaining robustness even when receiving input with reduced dimensionality.

5.2 Plant diseases recognition from digital images

The recognition and classification of leaf diseases of plants is a problem with many challenges to overcome. The analysis in identifying the diseases through the leaves can incur many false positives. For example, the symptoms of phytotoxicity are associated with some diseases due to similar leaf lesions.

In Abade *et al.* (2019) [12], we noticed that plant diseases are considered one of the main factors influencing food production, and to minimize losses in production, crop diseases must have fast detection and recognition. Before the advent of CNNs, traditional machine learning classification methods, such as SVM [106] and K-Means [107], were used to classify diseases in plants. Patil *et al.* (2011) [108] applied classic image processing technique for disease detection in sugarcane leaves by using threshold segmentation to determine leaf area and triangle threshold for lesioning area, getting the average accuracy of 98.60%. An approach developed by Singh *et al.* (2017) [109] uses genetic algorithms for image segmentation, which is crucial for disease detection in a plant leaf.

Relevant works approach to feature extraction techniques for the detection of plant diseases. It is possible to highlight the studies of Pydipati *et al.* (2006) [110], where there is the use of Color Co-occurrence Method (CCM) to determine whether texture-based hue, saturation, and intensity color features in conjunction with statistical classification algorithms could be used to identify diseased and normal citrus leaves under laboratory conditions. The leaf sample discriminant analysis using CCM textural features achieved classification accuracies of over 95% for all classes when using hue and saturation texture features. According to [111], a feature extraction is a promising approach capable of solving dichotomies between datasets constructed with images in controlled environments and images captured in the real world. This study proposed an ideal case approach in plant classification and recognition that was applicable in the real world and acceptable in laboratory conditions.

Due to the increase in processing capacity triggered by the use of GPUs [112], machine learning techniques have demonstrated significant accuracy in the classification and identification of plant diseases. These advances are demonstrated in the work of Rumpf *et al.* (2010) [106], which proposes an approach for the detection and differentiation of plant diseases using Support Vector Machine algorithms. In this study, the authors implemented a technique to identify beet diseases, in which depending on the type and stage of the disease, the classification accuracy was between 65% and 90%. Another approach based on leaf images and using Artificial Neural Networks as a technique for automatic detection and classification of plant diseases was used in conjunction with K-means as a clustering procedure proposed in the work of Hiary *et al.* (2011) [107]. On average, the accuracy of classification using this approach was 94.67%.

Thus, we proposed the use of M-CNNs, based on two distinct single architecture — AlexNet [3] and GoogLeNet (Inception v1) [4] —, to train and evaluate the PlantVillage dataset. Figure 5.2 shows a generic M-CNN architecture for this application.

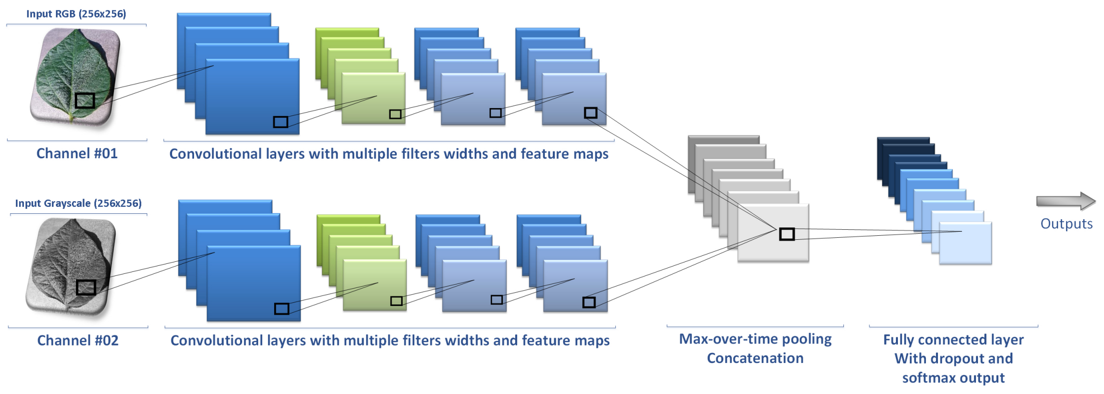

Input RGB (256x256)
Channel #01
Convolutional layers with multiple filters widths and feature maps
Input Grayscale (256x256)
Channel #02
Convolutional layers with multiple filters widths and feature maps
Max-over-time pooling
Concatenation
Fully connected layer
With dropout and
softmax output
Outputs

Figure 5.2: Generic M-CNN architecture. Taken from Abade *et al.*(2019) [12]. ©2019 IEEE.

As in Ferreira *et al.* (2018) [11], different versions of the original dataset were generated in an attempt to prove the contribution of these additional streams to the network performance. The extra streams consisted of a grayscaled and segmented version of the colored data and were combined as: **Version 1:** Color + Grayscale; **Version 2:** Color + Segmented and **Version 3:** Grayscale + Segmented.

Table 5.2: Mean F_1 score in percentage (%) of the proposed architectures. Table adapted from Abade *et al.*(2019) [12].

Model	**Dataset Type**	**Mean $\mathbf{F_1}$ Score**
AlexNet (from scratch)	Color	98.73
GoogleNet (from scratch)	Color	99.40
M-AlexNet (from scratch)	Version 1	**99.59**
	Version 2	99.20
	Version 3	99.23
M-GoogleNet (from scratch)	Version 1	99.55
	Version 2	99.38
	Version 3	99.41

It should be noted that knowing the gains of transfer learning techniques, we chose to train from scratch to demonstrate the possibility of customization and improvements in the learning process compared to a sample considered small for a dataset. Also, the additional inputs of the network provided better accuracy than the single stream CNN, showing that M-CNNs were able to enhance the general system, generating the best overall result in this work and keeping the mean F_1 scores regular and robust, independently of the chosen model.

Overall, we could conclude that an M-CNN model trained from scratch is better than a single channel model with transfer learning: faster convergence and reduced processing time. Furthermore, other image frequencies (e.g., grayscale) are crucial to improving the general accuracy.

5.3 Document classification using Bi-LSTM and ResNet-50

The Brazilian court system is currently the most clogged up judiciary system in the world since thousands of lawsuit cases reach the supreme court every day. These cases need to be analyzed to be associated with relevant tags and allocated to the right team. One of the first steps for the analysis is to classify these documents. In Braz *et al.* (2018) [113], this issue is addressed with the use of a Bi-LSTM model receiving as input the extracted text from the lawsuits PDF files.

Focusing on classifying five main types of legal documents handled by the Brazilian supreme court (STF) — documents that do not belong to these classes are grouped in a category called *Outros* —, over 1,000,000 text documents were manually labeled by a team of lawyers. This dataset has a high level of within-class diversity.

Furthermore, documents do not follow a pattern, not only in layout but also in terms of scanned image quality and whether or not they embed digital text or have only raster images. A significant amount of these documents contained handwritten annotations, stamps, stains.

Using the same dataset and single-stream Bi-LSTM model as [113], but with more samples, we achieved the results presented in Table 5.3 for text only.

Table 5.3: Classification report in percentage (%) using text as input.

	Precision	**Recall**	**F1-score**	**Support**
Acórdão de 2 instância	42.03	97.82	58.80	2336
Agravo em recurso extraordinário	53.35	82.02	64.65	13811
Despacho de admissibilidade	51.32	95.63	66.80	1970
Outros	98.81	90.86	94.67	416680
Petição do RE	61.15	87.27	71.91	36693
Sentença	61.18	96.21	74.80	9835
Average	93.40	90.50	91.38	481325

Nonetheless, we checked if only the image information was enough to classify the legal documents. Using a ResNet-50 approach, we saw that even with fine-tuning and hyperparameter optimization, the imbalance of the dataset was too much to handle. With a high accuracy as a result of a large number of *Outros*, the other metrics were weak.

Thus, to combine two domains aiming for a performance gain, the previous ResNet-50 was fused to the original Bi-LSTM. The ResNet dealt with the PDF image files, while the Bi-LSTM learned the text features. Insisting on image features was an empirical feeling that we thought it would benefit the network. The M-CNN results are shown in Table 5.4.

Table 5.4: Classification report in percentage (%) using text and image as input.

	Precision	**Recall**	**F1-score**	**Support**
Acórdão de 2 instância	94.63	95.85	95.24	2336
Agravo em recurso extraordinário	95.57	83.48	89.12	13811
Despacho de admissibilidade	95.38	81.83	88.09	1970
Outros	98.76	98.97	98.86	416680
Petição do RE	91.17	93.34	92.24	36693
Sentença	94.10	96.59	95.33	9835
Average	97.96	97.96	97.94	481325

Although the results with only text were already excellent, we can see that the inclusion of image features, which alone did not work, were essential for gain in accuracy. Thus, we expand this concept and present it in the following Section.

5.4 Sequence-aware multimodal page classification of Brazilian legal documents

Still dealing with the problem of the overflow of cases in the Brazilian judicial system, we explored multimodal classification of documents from Brazil's Supreme Court. In here, we used a subset of VICTOR dataset, named *small VICTOR*. Each lawsuit in this dataset is an ordered sequence of pages, stored both as an image and as a corresponding text extracted through optical character recognition. By combining these two different sources of information, previously extracted from individual classifiers – ResNet-50 [18] for image classification and a stack of convolutional neural networks for text classification –, we create a fusion module that is capable of handling absent textual or visual inputs by using learned embeddings for missing data. Table 5.5 presents a simplified F_1 measure, in $\%$, comparison between our test set.

Several other approaches were implemented before reaching the final fusion module that concatenates each modality embedding followed by fully connected layers. Primarily, the combination of raw images with raw text that was the same strategy used in the aforementioned paper [113] was tested. However, since we had a distinct subset with more refined data, the results were not as good as the previous one and could not solve the same issue. Images were too heavy to load, and we decided to resize them to fit into the model. This spatial reduction leads to a good amount of information loss, resulting in a bad image classifier and thus of little help to the text classifier, regardless of the fusion strategy chosen. We collected image input activations to feed the image network to minimize this issue, leading to a more promising model.

Furthermore, changing the artificial intelligence framework also increased our performance as it seemed that the combination of PyTorch [114] and fastai [115] had intrinsic optimizations that improved our Tensorflow [95] + Keras [94] multimodal fusion models. The latter had trouble converging with several combinations of parameters.

Continuing the process of classification improvement using the fusion module, the order of the pages was also part of the solution: sequence classification techniques as a conditional random field (CRF) [116] were used to output a sequence of class predictions.

An additional learning in this work was the implementation of the LayoutLMv2 network [117]. This architecture is designed to deal with the same multimodality that we proposed, utilizing document layout and its OCR as the leading reference on the document + text issue. Using Transformer encoders on each stream, this model also contains a spatial-aware self-attention mechanism into the Transformer architecture, understanding the relative positional relationship among different text blocks.

Table 5.5: F_1 measures in percentage ($\%$) for text only, image only, the fusion module, the fusion module with sequence classification and the reference LayoutLMv2 model.

	Text	**Image**	**FM**	**FM + Sequence classification**	**LMv2**
Acórdão de 2 instância	89.96	18.45	90.74	88.97	60.13
Agravo em recurso extraordinário	55.72	11.33	57.92	61.16	23.91
Despacho de admissibilidade	62.94	08.44	63.98	64.07	23.72
Outros	97.31	61.72	97.24	97.46	96.54
Petição do RE	75.59	32.59	75.47	79.67	72.25
Sentença	80.53	43.52	82.04	85.26	66.68
Average	77.01	29.34	77.90	79.43	57.21

Chapter 6

Proposed topology

M-CNNs have been developed, applied and used in many situations and applications nowadays [6, 12, 27, 28, 29, 30, 31, 32, 33]. This model architecture is derived from traditional CNNs proposed by [2] and allow to employ basically all, traditional (or not) models available in the literature, as *LeNet*[2], *AlexNet*[10], *VGG*[17] and many others models, basically adjusting (or modifying) the fusion stage[6].

Usually, works that address applications issues to a M-CNN approach do not focus on the fusion stage (e.g. [12, 17]). However, there are some efforts [27, 118, 119, 87] in order to increase the performance of the networks. Gamulle *et al.* [27] and Tu *et al.* [118] uses a multi-stream approach in order to recognize human actions from video sequences. The former focuses on learning salient spatial features and mapping their temporal relationship with the aid of Long-Short-Term-Memory (LSTM) networks and uses two different fusion methods: averaging and training a multi-class linear SVM using the softmax scores as features. The latter construct an appearance and a motion stream, concatenating the streams in a fusion module based on Spatio-temporal 3D convolutions. Azar *et al.* [119] use M-CNNs to recognize group activities and use concatenations to fuse all its streams. In [87], M-CNNs are used to classificate high-resolution aerial scenes, and two fusion techniques are evaluated, concatenation and addition.

Karpathy *et al.* [6] addressed the fusion issue to a novel model, placing two separate single-frame networks time-delayed apart and merging their outputs in a fusion step, allowing improvements in the accuracy scores on video action recognition tasks. As described in Feichtenhofer *et al.* [28], the fusion stage can be performed at a convolutional layer without loss of performance (accuracy). Based on this finding, we propose in this manuscript a novel model based on a new cross-fusion method applied for all available models of M-CNNs. This new proposed cross-fusion method is focused on observed features, specifically in the *ReLu's* stage, used by many of traditional M-CNNs models developed in all known approaches (e.g., in [6, 8, 28]).

Our proposed model is based on a crossing signal inference among each data streaming (or channel) output from the convolution stage and processed by the ReLU's stage, connecting each of this output with others outputs coming from all others data streamings (or channels) using a new fusion function approaching. This model is formally described in the following Section.

6.1 Lattice Cross-fusion Strategy

In system analysis, there is a field that studies the enhancement or restoration of degraded signals [120]. Inspired by the fact that there is no signal processing without degradation and that a CNN signal suffers loss along its course, we decided to apply basic signal operations between activation layers. Especially the ones that use the ReLU (Rectified Linear Unit) function: $g(z) = max\{0, z\}$, is a non-linear function used by many classical M-CNNs models developed in several popular approaches (e.g. in [6, 8, 28]). ReLU outputs zero across half its domain, making the derivatives through it remain significant whenever the unit is active. Nonetheless, they cannot learn with gradients near zero [1].

To boost up near zero gradients and get a hold of important features that may be left out, we combine two different signal streams and a crossing signal inference of the operation result to the input of the subsequent layer. Equation 6.1 presents a fusion $F(s)$ of two signals $s, \bar{s}$, where $\odot$ represents the chosen mathematical operation.

$$F(s, \bar{s}) = s \odot \bar{s}. \tag{6.1}$$

Then, a layer L can be defined as:

$$\begin{aligned} L(s, \bar{s}) = [&F_a(C_a(s), C_b(\bar{s})), \\ &F_b(C_b(\bar{s}), C_b(s))]. \end{aligned} \tag{6.2}$$

As described in Almeida *et al.* (2019) [91], the cross-fusion function $f : C_a, C_b, ..., C_k, ...C_n \rightarrow y$ combines the $n-1$ convolutional layers with the $C_k \in \mathbb{R}^{h \times w \times d}$ where h, w and d are the height, width, and depth (number of channels/streams), respectively.

The cross-fusion general module is defined in Equation 6.2. It computes the operation $\odot$ of two convolutional layers inputs, C_a and C_b, connecting the result as an input of the next layers $C_{a'}$ and $C_{b'}$. It is important to point out that any mathematical operation can be applied in the $\odot$ stage, such as addition, subtraction, average.

These fusion modules are repeated along with all CNNs' *convolution-ReLU* layers sets, finishing with a late fusion process right before the fully connected stack step. Figure 6.1 presents a visual definition of this proposed cross-function strategy.

The signal crossing can be noticed in the visualization of the L-strategy module found in Figure 6.2.

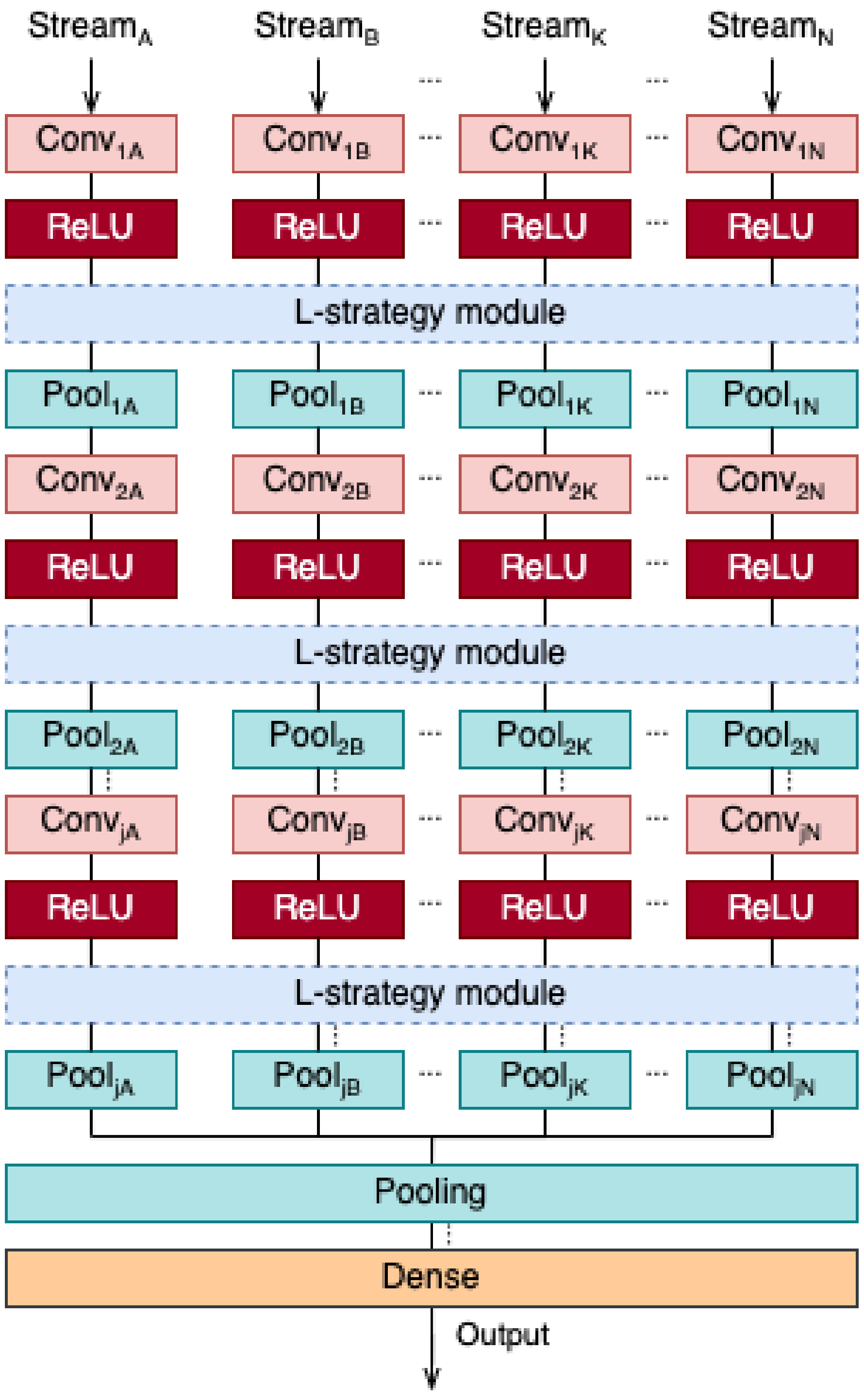


Figure 6.1: A general L-CNN model. Convolutional layers are represented by the color red, followed by a wine color ReLU indicator, while the fusion modules are dotted-blue. Pooling layers are expressed by the color green. Other layers are included for an architecture disclosure.

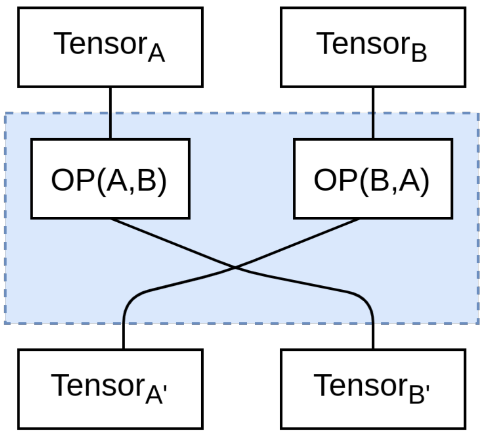


Figure 6.2: A closer look at the L-strategy module. OP represents a mathematical operation.

This signal enhancement makes it possible to empower CNNs with fewer layers and use simpler structures to achieve good results as state-of-the-art networks with tons of convolutional layers modules. This structure cannot run in modest hardware. The following Section shows our work methodology, and we demonstrate that our results, even with insignificant streams, generally outperform classical late fusion approaches.

6.2 Experimental Evaluation

To evaluate our approach, our cross-fusion function is set to an *average* operation and the used baseline (single and dual-stream) architecture is AlexNet [3].

An average fusion is defined as $y^{\text{avg}} = f^{\text{avg}}(C_a, C_b, ..., C_k, ..., C_n)$. It computes the average of the n convolutional layers, connecting as an input of the next pooling layer.

Our new AlexNet-LCNN is defined as *C(96,11,4)* $\rightarrow$ *LF(average)* $\rightarrow$ *P(2)* $\rightarrow$ *C(256,11,1)* $\rightarrow$ *LF(average)* $\rightarrow$ *P(2)* $\rightarrow$ *C(384,3,1)* $\rightarrow$ *C(384,3,1)* $\rightarrow$ *C(256,3,1)* $\rightarrow$ *LF(average)* $\rightarrow$ *P(2)* $\rightarrow$ *FL* $\rightarrow$ *FC(4096)* $\rightarrow$ *D(0.4)* $\rightarrow$ *FC(4096)* $\rightarrow$ *D(0.4)* $\rightarrow$ *FC(10)*, such that *C(d, f, s)* indicates a convolution layer with d filters of spatial size $f \times f$, applied to the input with stride s. *LF(α)* means the lattice fusion realized with an operation cross-fusion function. *P(s)* is the pooling layer with stride s. *FL* is a layer that flattens the input. *FC(n)* is a fully connected layer with n nodes. *D(p)* is a dropout layer of p as a dropout rate, used exclusively during the training step.

To compare performances, we also implemented a late fusion multistream AlexNet (AlexNet-MCNN), consisting of two independent streams that merge right before the first fully connected layer as highlighted in [6].

Additionally, for comparison proposes, we also evaluated a cross-modal CNN (AlexNet-XCNN) [8], which is a typically image-based approach that each of the stream image receives its own CNN super layer, with cross-connections inserted after the pooling operation, and total weight sharing in the fully connected layers. This model was developed to explore the crossing-signal inference, and its accuracy performance will be used to compare with our novel proposed L-CNN model. Results also considered the traditional single-stream AlexNet architecture [3] for both of our streams, described below.

The chosen testbed data set to evaluate our model was the CIFAR-10 [10], a popular image classification benchmark data set. For a multistreaming scenario, the selected streams were purposely worsened to evaluate the robustness increased by the proposed L-CNN model — the main goal is not to achieve or improve state-of-the-art results. In the first stream, a grayscaled version of CIFAR-10 is used, and in the second stream, an edge extraction — created with Canny edge detector [121] — of the first stream, presented respectively in Figures 6.3 and 6.4. To respect AlexNet's original first convolutional stage constraints, both input streams were resized to 224×224.

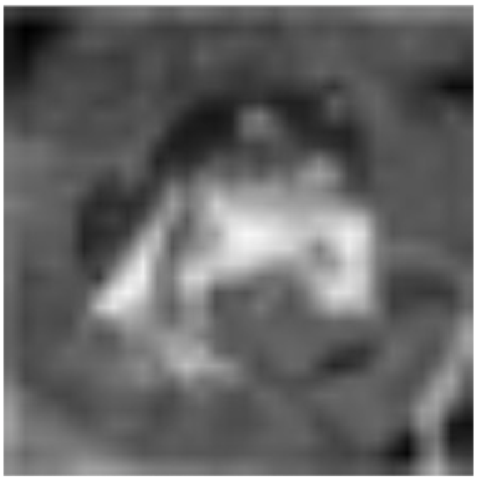


Figure 6.3: Grayscale image.

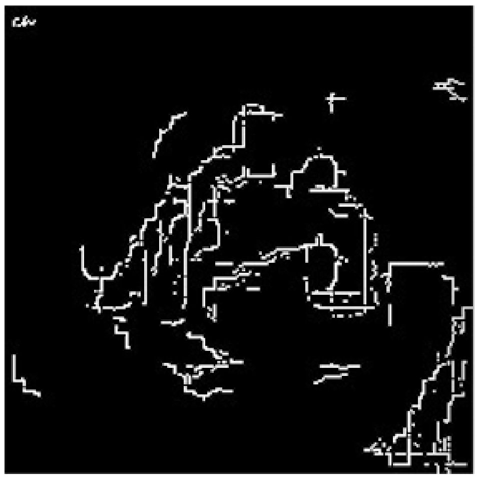


Figure 6.4: Image with edge detection.

Figure 6.5: A class sample from the CIFAR-10 data set [10]. In (a) the first stream is a 224×224 grayscale image. In the second stream (b), the same image with edge detection.

Table 6.1 presents all the achieved accuracies with the previously described streams and model comparisons when trained for 260 epochs. This number of epochs was chosen accordingly to loss and accuracies graphs presented in [8]. Given that the edge detection single stream shows constantly poor results in accuracy and loss, we can describe this stream as a distractor to the network. Even the grayscale stream did not perform well in a single stream scenario, as described in Table 6.1. It also can be noticed that the L-CNN method outperforms all the others approaches, including the single-stream models, using two low-quality streams and fewer features to learn, considering that we took the color information that would support the network in feature learning, in according to [1].

The graphs shown in Figures 6.6 and 6.7 confirm that our edge detection signal does not add consistent information to the network, as expected. Furthermore, the grayscale stream does not have a good performance by itself. Using this information, we can also point out the stability from the AlexNet-LCNN, achieving its peak of about 50 epochs and maintaining

Table 6.1: CIFAR-10 data set accuracies in percentage (%) using a 2-streaming schema.

Model	Loss	Accuracy
AlexNet (grayscale)	3.12	16.64
AlexNet (edge detection)	12.91	10.08
AlexNet-MCNN	2.92	48.53
AlexNet-XCNN	2.35	48.96
AlexNet-LCNN	**2.92**	**62.57**

accuracy and loss during all the training process, unlike AlexNet-XCNN and even AlexNet (grayscale). Moreover, the lattice module increased the robustness of the network results using poor signals as streams.

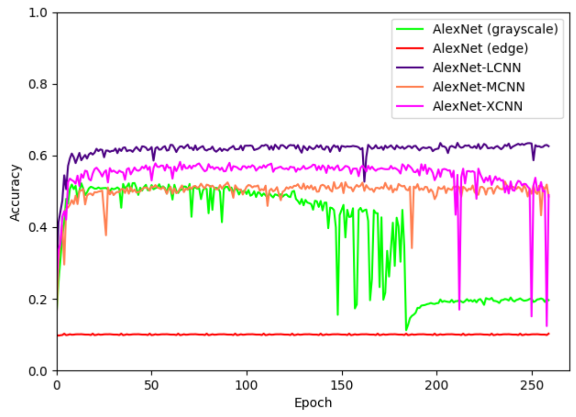


Figure 6.6: CIFAR-10 test accuracies.

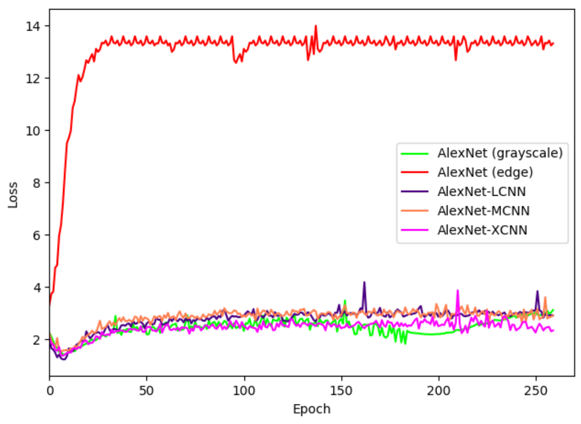


Figure 6.7: CIFAR-10 test losses.

Figure 6.8: CIFAR-10 test accuracies and losses under training epochs.

6.3 Turning old models fashion again: Recycling classical CNN networks using the Lattice Transformation

In our test expansion of the previous technique, two additional mathematical operations to the average were implemented in our fusion module: addition and subtraction. Further, we adapted 9 different architectures' backbones to demonstrate that the lattice strategy has significant improvements with varying kinds of structures, varying in-depth, the number of convolutional layers, and parameters in general. Full models can be found in the Appendix. With the exception of the LFR-CNN model that used the Adam optimizer at a learning rate of 0.00001, all experiments were performed using a default SGD optimizer with a 0.01 learning rate.

Initially, three sets of data were used to evaluate our fusion strategy: the previously used CIFAR-10 [10], CIFAR-50, and CIFAR-100 [122]. Different sizes of datasets were chosen to analyze our strategy performance with a distinct amount of data samples. To create images for the second stream, we generated a mirrored input using an edge extraction created with

Table 6.2: Network parameters by implementation type.

Architecture	Type	Parameters
Alexnet	Single stream	23.98M
	Multistream - late fusion	30.08M
	Multistream - lattice	30.08M
ResNet-18	Single stream	11.19M
	Multistream - late fusion	22.38M
	Multistream - lattice	34.93M
ResNet-34	Single stream	21.31M
	Multistream - late fusion	42.62M
	Multistream - lattice	65.29M
ResNet-50	Single stream	23.59M
	Multistream - late fusion	47.18M
	Multistream - lattice	57.33M
DenseNet-121	Single stream	70.47M
	Multistream - late fusion	14.09M
	Multistream - lattice	14.08M
DenseNet-169	Single stream	12.65M
	Multistream - late fusion	25.31M
	Multistream - lattice	25.30M
DenseNet-201	Single stream	18.34M
	Multistream - late fusion	36.68M
	Multistream - lattice	36.66M
Xception	Single stream	20.88M
	Multistream - late fusion	41.76M
	Multistream - lattice	41.72M
FR-CNN	Single stream	53.29M
	Multistream - late fusion	—-
	Multistream - lattice	99.55M

a Canny edge detector [121], as in Almeida *et al.* (2019) [91]. Also, it is important to emphasize that all images were resized to a 224×224 shape with the purpose that they fit most of our networks' default input shapes.

With the need to expand the tests, however, it became necessary for us to include other data sources with different purposes. For image classification, NORB; for object detection, BDD100K. Since the NORB data already has two different streams to fill in our network specifications, as stated in Chapter 4, the only modification needed was on the BDD100K: we apply the same approach used on the CIFAR datasets, an edge extraction illustrated on Figure 6.9.

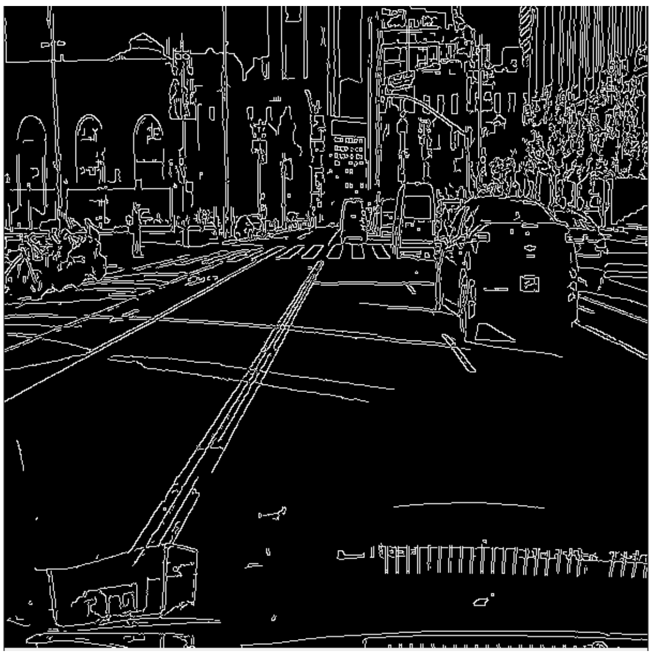


Figure 6.9: A frame of the BDD100K dataset with Canny edge detection.

6.3.1 Hardware and training time

Most networks were adapted from their previous implementation on the Keras library [94]. AlexNet and FR-CNN were based on their definition papers. Fusion operations were created using tensor operations available in the Tensorflow framework [95]. The experiments were trained in one RTX 3090 GPU, and all experiments, including the NORB and BDD10K dataset, took approximately 6 months.

6.3.2 Results

The CIFAR datasets had their training set subsampled in a cross-validation procedure with 5 folds. The following accuracy results are presented by fold. No data augmentation nor transfer learning was used in the whole process. Due to many experiments, we chose to gather all accuracies in one figure. In all plots, the color *blue* will represent the late fusion version of the network. Colors *orange, green*, and *red* are the L-fusion architectures with average, sum, and subtraction operations.

Considering the increase of the network parameters presented in Table 6.2, but also an accuracy gaining of 5.66% using a default L-strategy average operation over a simple late fusion technique, independently of the given dataset or dataset quality, it is fair to say that the occasional increase in parameters is outweighed by the performance boost. Our losses graph, Figure 6.41, also shows that the lattice-adapted networks tend to converge faster.

The goal of this work is not to beat any kind of state-of-the-art models, as previously noted. Thus, it is remarkable to achieve strong accuracies with very degraded streams. Our L-Xception with the average operation, for example, reached an 87.02% accuracy on CIFAR-10.

Also, using the same dataset as a comparator, a late fusion AlexNet went from 42.69% to 58.66%. AlexNet has a very straightforward backbone, is easy to implement, and can run smoothly on plain hardware.

Tables 6.3, 6.4, and 6.5 present the mean accuracy of the folds for all models and datasets tested here, being a summarized version of the graphs presented in Figures 6.13, 6.23, 6.27, and 6.37.

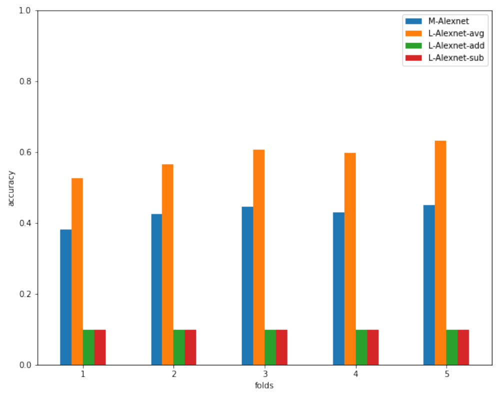


Figure 6.10: AlexNet - CIFAR-10

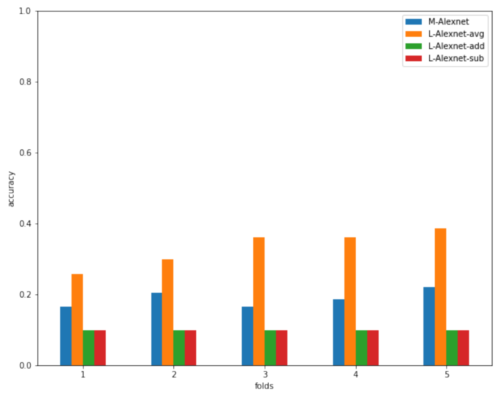


Figure 6.11: AlexNet - CIFAR-50

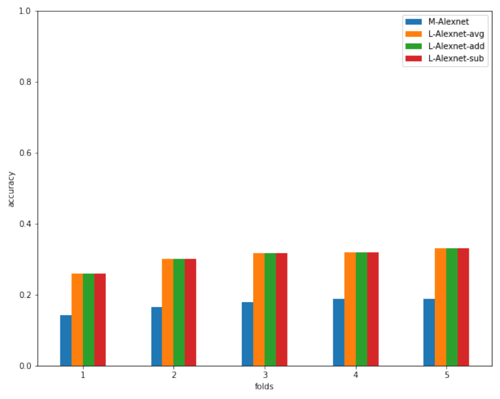


Figure 6.12: AlexNet - CIFAR-100

Figure 6.13: Compilation of accuracy comparisons using AlexNet architecture variations.

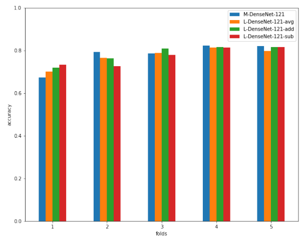


Figure 6.14: DenseNet-121 - CIFAR-10

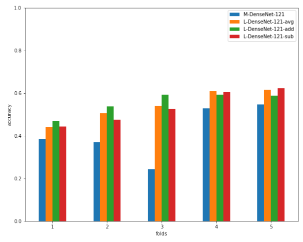


Figure 6.15: DenseNet-121 - CIFAR-50

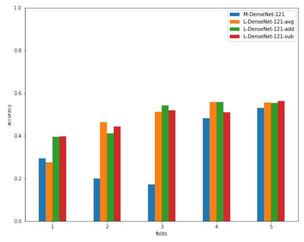


Figure 6.16: DenseNet-121 - CIFAR-100

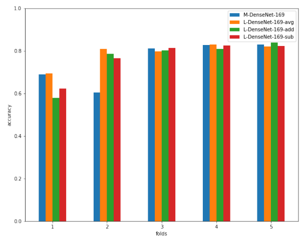


Figure 6.17: DenseNet-169 - CIFAR-10

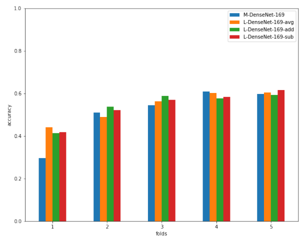


Figure 6.18: DenseNet-169 - CIFAR-50

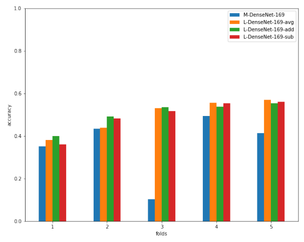


Figure 6.19: DenseNet-169 - CIFAR-100

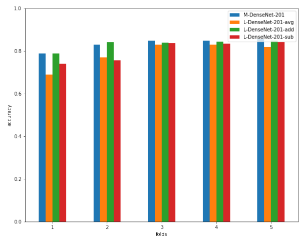


Figure 6.20: DenseNet-201 - CIFAR-10

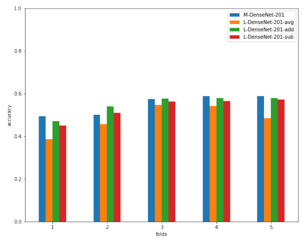


Figure 6.21: DenseNet-201 - CIFAR-50

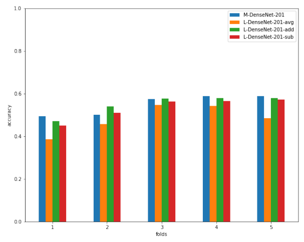


Figure 6.22: DenseNet-201 - CIFAR-100

Figure 6.23: Compilation of accuracy comparisons using DenseNet architecture variations.

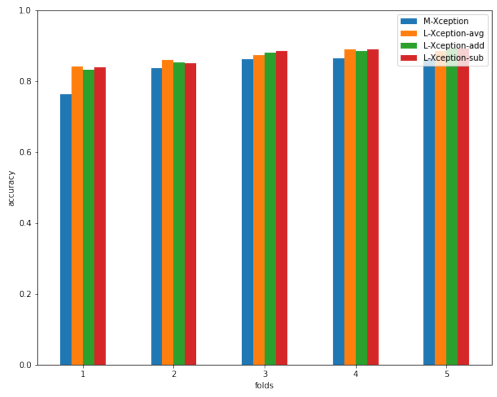


Figure 6.24: Xception - CIFAR-10

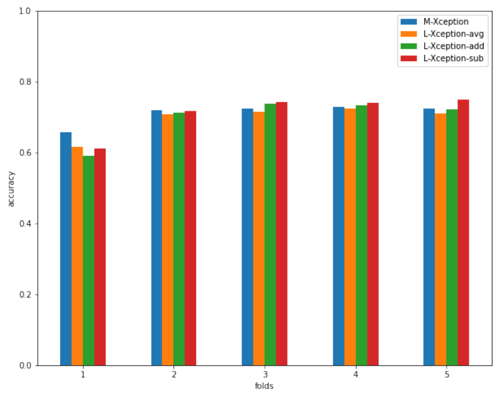


Figure 6.25: Xception - CIFAR-50

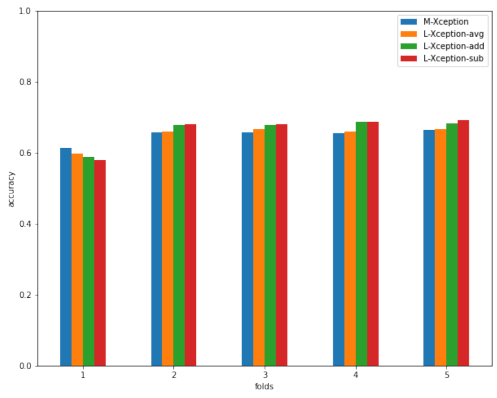


Figure 6.26: Xception - CIFAR-100

Figure 6.27: Compilation of accuracy comparisons using Xception architecture variations.

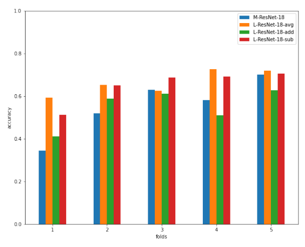


Figure 6.28: ResNet-18 - CIFAR-10

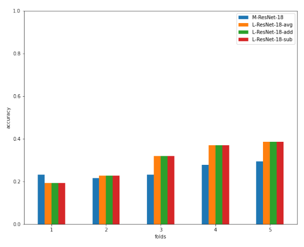


Figure 6.29: ResNet-18 - CIFAR-50

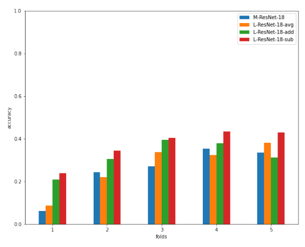


Figure 6.30: ResNet-18 - CIFAR-100

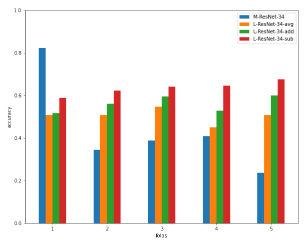


Figure 6.31: ResNet-34 - CIFAR-10

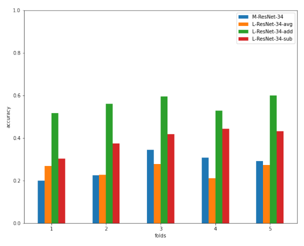


Figure 6.32: ResNet-34 - CIFAR-50

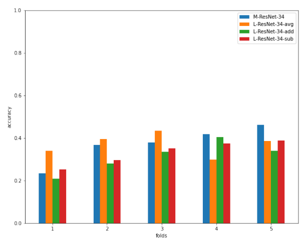


Figure 6.33: ResNet-34 - CIFAR-100

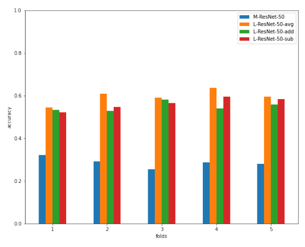


Figure 6.34: ResNet-50 - CIFAR-10

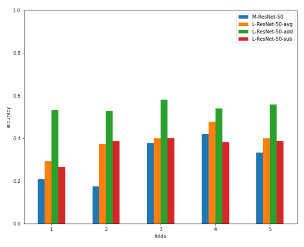


Figure 6.35: ResNet-50 - CIFAR-50

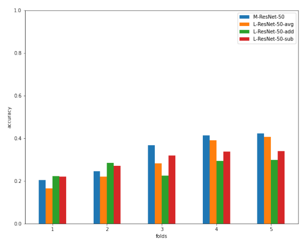


Figure 6.36: ResNet-50 - CIFAR-100

Figure 6.37: Compilation of accuracy comparisons using ResNet architecture variations.

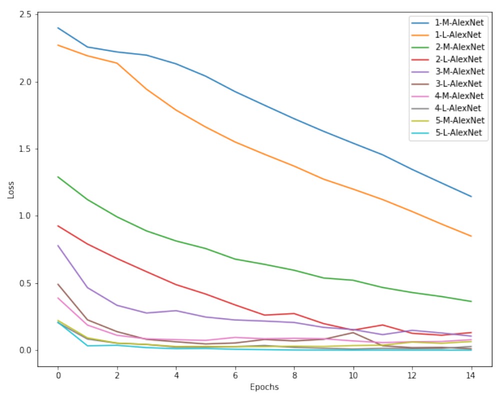


Figure 6.38: AlexNet losses per fold - CIFAR-10 dataset

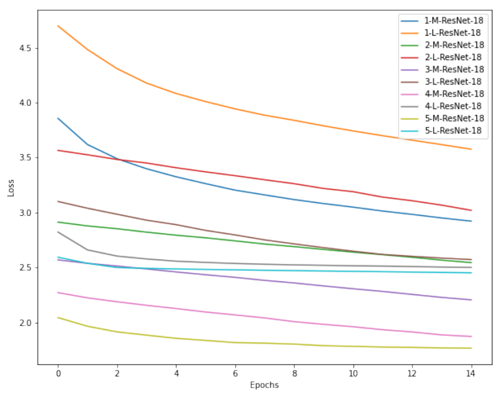


Figure 6.39: ResNet-18 losses per fold - CIFAR-10 dataset

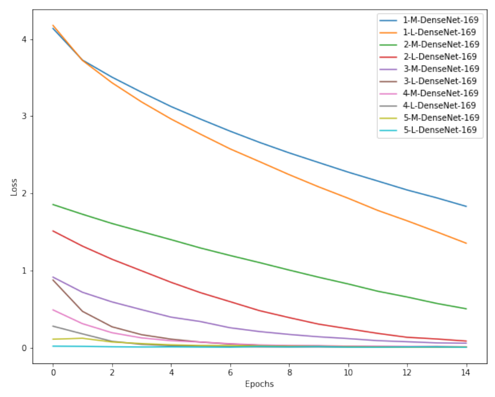


Figure 6.40: DenseNet-169 losses per fold - CIFAR-100 dataset

Figure 6.41: Loss comparison per fold showing different architectures and distinct streams.

Table 6.3: Mean accuracies in percentage (%) of all trained models for CIFAR-10.

Architecture		**AlexNet**	**DenseNet-121**	**DenseNet-169**	**DenseNet-201**	**Xception**	**ResNet-18**	**ResNet-34**	**ResNet-50**
M-CNN		42.69	77.93	75.26	**83.39**	83.79	55.55	44.06	28.74
L-CNN	*avg*	**58.66**	54.22	54.04	57.28	87.02	**66.34**	50.45	**59.53**
	add	10.00	**78.486**	76.31	83.20	86.89	55.01	56.05	54.87
	sub	10.0	77.40	77.06	80.19	**87.17**	64.96	**63.49**	56.31

Table 6.4: Mean accuracies in percentage (%) of all trained models for CIFAR-50.

Architecture		**AlexNet**	**DenseNet-121**	**DenseNet-169**	**DenseNet-201**	**Xception**	**ResNet-18**	**ResNet-34**	**ResNet-50**
M-CNN		18.85	35.74	35.95	54.93	71.12	25.06	27.37	30.30
L-CNN	*avg*	**33.6**	47.39	49.60	48.39	69.46	29.90	25.22	**38.96**
	add	10.00	**55.66**	54.18	**60.30**	69.96	37.63	37.84	34.48
	sub	10.00	53.47	**54.21**	57.60	**71.22**	**39.00**	**39.48**	36.50

Table 6.5: Mean accuracies in percentage (%) of all trained models for CIFAR-100.

Architecture		**AlexNet**	**DenseNet-121**	**DenseNet-169**	**DenseNet-201**	**Xception**	**ResNet-18**	**ResNet-34**	**ResNet-50**
M-CNN		17.30	**77.93**	75.26	**83.39**	64.98	25.36	**37.18**	**30.94**
L-CNN	*avg*	**30.58**	77.30	**79.05**	78.77	65.02	27.03	37.11	29.32
	add	10.00	49.23	50.43	54.92	**66.29**	32.08	31.38	26.55
	sub	10.00	48.72	49.51	53.23	66.39	**37.09**	32.44	29.73

Table 6.6: NORB accuracies in percentage (%) on the test set.

Architecture		**DenseNet-169**	**ResNet-50**	**AlexNet**
Single stream - left image		60.21	20.00	20.00
Multistream - late fusion		37.27	67.85	20.00
Multistream - lattice	*average*	**70.91**	46.82	20.00
	addition	52.53	31.38	20.00
	subtraction	51.56	**75.58**	**83.21**

After the experiments with variant and degraded CIFAR datasets, we decided to understand our proposal better by applying our method in a real and not overused dataset. Table 6.6 presents a comparison between three different-sized architectures in three forms: single stream, multistream with late fusion, and multistream with our lattice cross-connection. Results show that at least one lattice operation could outperform all other results, demonstrating the flexibility of the technique when we come across not-so-good results.

The BDD100K and the LFR-CNN bring complexity and a challenge to our strategy. On the network aspect, we are dealing with two different outputs with distinct goals and implementations. On the data side, we have a very difficult in-the-wild dataset that is completely outside the scope of what we were evaluating the models: here we have actual driving scenes.

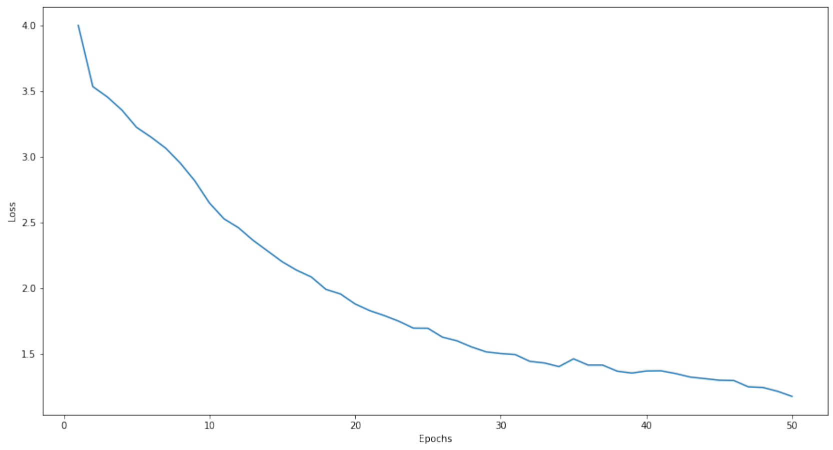


Figure 6.42: Training losses of the BDD100K dataset.

Using the work of Bhargava (2019) [123] as the baseline reference, since it uses the same model and backbone as ours, we report a mAP@75 improvement of 50.16%, as shown in Table 6.7. This improvement was obtained after training per 50 epochs without loading pre-trained weights or parameter optimization.

A closer look at Figure 6.42 shows that the loss drop tends to diminish if trained for a longer period. However, since we had computational limitations, 50 epochs were our maximum effort.

Table 6.7: BDD100K mAP@75 in percentage (%) on the validation set.

Architecture	**mAP@75**
FR-CNN	18.20
LFR-CNN	27.30

6.4 Discussions

The first applications presented here showed a more straightforward way of constructing M-CNNs. Simple network constructions for complex applications. The results all seemed to improve a single approach, leading to the hypothesis that networks with multiple entries were naturally good candidates for solving different problems. The inclusion of features proved real.

With the proposition of the L-strategy, we start to raise the level of our efforts. The lattice fusion comes directly with a new modeling structure as we begin to customize more previously consolidated networks. Primarily, we used an AlexNet architecture as a baseline and CIFAR-10 data set and implemented three different model versions: a multistream CNN with late fusion, a cross-CNN, and our lattice-CNN, alongside two single streams of traditional AlexNet. Experimental results show that the proposed LF outperformed all the models above at least 28% compared to all tested models. Also, the proposed fusion demonstrated robustness and stability, even when distractors were used as streams.

With the expansion of the test, we used other different backbones and 4 datasets (including the CIFAR-10 mentioned above) to demonstrate the robustness of the proposed technique. Our experimental results show that the proposed strategy has faster convergence and flexibility of operational switches.The previously noted stability was maintained.

Considering that we stayed in a safe field during all the anterior applications, we managed to elevate our strategy to a new complexity: object detection. Here, we have to not only classify, but we also have to locate an object during a video. That is a task that requires two outputs and another it is indeed another layer of difficulty. Nevertheless, the L-strategy still performed well with the restricted training conditions which were our initial assumptions of not using pre-trained weights, not recalibrating the hyperparameters and not doing fine tuning.

With a gain of approximately 50% we are positive that the suggested implementation can go even further.

6.4.1 What if L-strategy does not improve my results?

There is a possibility that the described module and its default operators will over-amplify signals in a way that the model will simply not learn. Using signal processing knowledge, we know that when this kind of peaks occurs, we can perform a compression to approximate low and high signals. Therefore, we propose using a logarithm-based compression function that will balance the broad range, reducing the difference between large and small values. This function is described in Equation 6.3, where s and $\bar{s}$ are the incoming signal streams.

$$F(s, \bar{s}) = s - \log(1 - \bar{s}). \tag{6.3}$$

An example is our L-VGG-16 [17] implementation. Our first run using an *average* function on CIFAR-10 went well, but not enough to beat the M-VGG-16 architecture. Further runs with other defined operators were boosting both tensor streams, and our model found its local minimum very fast, without room for optimizations. Figure 6.43 shows a comparison between operations and modeling type.

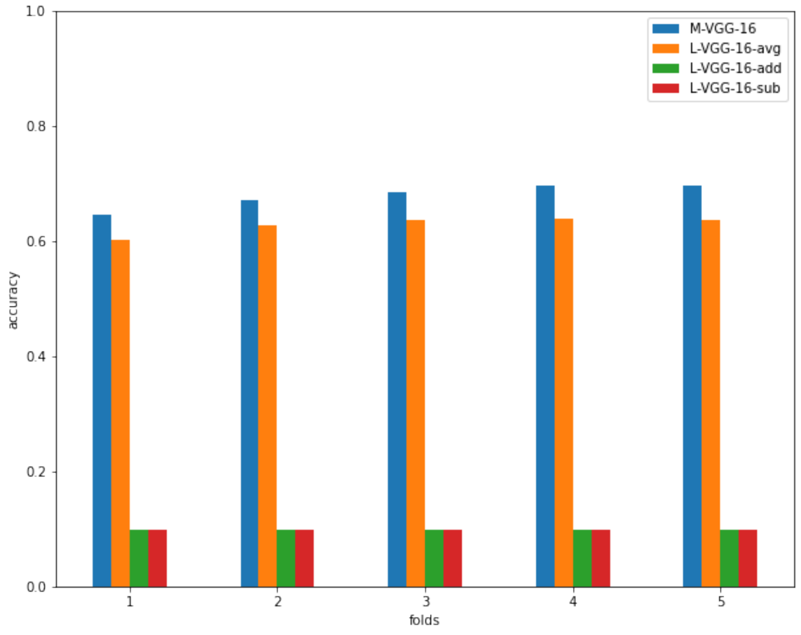


Figure 6.43: VGG-16 results on CIFAR-10.

Table 6.8 presents all obtained VGG-16 accuracies for each fold on the test set. We can clearly see that our log-compression function improved the overall results.

Table 6.8: VGG-16 accuracies in percentage (%) on the test set of CIFAR-10 for each fold.

Fold	**Architecture**				
	M-VGG-16	**L-VGG-16-avg**	**L-VGG-16-log**	**L-VGG-16-add**	**L-VGG-16-sub**
1	64.54	60.21	**72.65**	10.00	10.00
2	67.20	62.80	**74.26**	10.00	10.00
3	68.47	63.80	**75.62**	10.00	10.00
4	69.63	63.97	**75.94**	10.00	10.00
5	69.66	63.77	**75.55**	10.00	10.00
Average	67.90	62.91	**74.80**	10.00	10.00

Notice that the technique mentioned above was meant to be used with all default hyperparameters. Still, it does not work for all situations. A VGG-19 with standard initialization will continue to have its signal overamplified, converging too quickly. Nonetheless, basic optimization approaches are sufficient to train the network smoothly gathered all accuracy: the learning rate, for example, controls the adaptability of the dataset to the network and affects directly in weights updated. Hence, when faced with large amplifications caused by the L-strategy, it is usually enough to decrease the starting learning rate or apply schedulers. To illustrate this, we present in Table 6.9 a comparison between a multistream late fusion approach and the L-strategy approach with the non-default learning rate of 10^{-4}, for as much as the regular parameterization generated unusable results.

Table 6.9: VGG-19 accuracies in percentage (%) on the test set for each fold.

Fold	**Architecture**			
	M-VGG-19	**L-VGG-19-avg**	**L-VGG-19-add**	**L-VGG-19-sub**
1	66.63	56.43	65.11	**66.71**
2	66.25	50.64	69.52	**70.52**
3	69.32	52.52	71.21	**71.84**
4	69.11	54.85	**72.89**	72.45
5	64.55	55.53	73.38	**73.49**
Average	67.17	53.99	70.42	**71.00**

Finally, we can observe that the presented results in this Section are not linear. There is not a definitive better fusion operation. Although we recommend using *average*, we displayed our results using a trial-and-error method.

Chapter 7

Conclusions

This work explores how multistream deep neural networks are a gamechanger in the deep learning field. However, they have not been extensively researched yet. We came across many gaps in this kind of building during our study sessions: where is the best point of fusion of several equal or different backbones? What techniques can we use to get the best result from this inclusion of features? Do all backbones work the same when it comes to fusing them?

With the L-CNN model, we could at least clarify some of these questions. Using signal processing fundamentals, we developed a general module to get the best that multiple signals could offer us by overamplifying our inputs and crossing them to serve as the new input to the sequential convolutional layer.

Nonetheless, we also observed that when the strategy fit well the backbone, the results were likely to be more expressive than the single versions or the late fused backbones regardless of the final application. This observation included elementary structures, such as AlexNet, or previously considered state-of-the-art networks. The expressiveness led to an exciting conclusion.

Regarding our predefined hypothesis 1 in Chapter 1, we could see that our L-strategy can give extra life to obsolete models, offering an uncomplicated module design to basic structures, allowing them to become competitive again.

This model recycling is especially interesting when the available hardware capacity is not enough to train the super complex models that come out almost daily. Further, no one will not have to deal with constructing a whole new stack, it is just a matter of rehashing networks that are already known. Thus, our goal to reuse previous state-of-the-art architectures with few modifications to keep them in the game was successfully achieved.

Still, there is no definitive technique. The general model turned out not to be suitable for certain types of architectures. We showed that we could obtain an increase of accuracy up to 63.21% in image classification tasks using an easy-implemented network and a boost of 50.16% on the mAP baseline of an object detection challenge with a classical model structure, nonetheless, the L-strategy with very large kernel sizes tends to overamplify so much the signal that the default hypermeters are not enough to train a good model, causing the process to get stuck right on the beginning of the training cycle.

7.1 Future Work

Besides dealing with the over-amplification issues, future work will also include new steps exploring improvement categories: the next move is to enhance a range of other models, following an exploratory line of research for new techniques to give extra life to models becoming obsolete.

Additionally, we intend to explore not only CNNs but also every kind of structure that includes a ReLU activation, such as Long short-term memory (LSTM) and Generative Adversarial Networks (GAN), evaluating the proposed technique in other data modalities. Different application tasks are also considered when we explore distinct types of architectures.

Since this module presents implementation flexibility, we plan to test the number of fusion modules needed to improve the final network representation, considering that not all ReLU activations need to be crossed. Also, on-training mixed operations to check which one effectively boosts the signal in the right way is a future research topic.

Finally, among the improvement and development activities, the need to create an automated strategy for the L-CNN conversion of existing models, including those in disuse, is worth mentioning because more advanced models and efficient architectures have superseded them. Currently, the process is partially automated, which is dependent on the development library, in this case, Tensorflow[95]. It is worth mentioning that this is also a possible obstacle to the wide dissemination and adoption of the L-CNN conversion technique by other researchers. Therefore, it is necessary to contemplate the support to other libraries adopted by the scientific community regarding the development to be carried out in the future. This approach is crucial for expanding the different models and inspiring the new analysis of changes in the signal flow of the other architectures and existing models.

7.2 Publications

Table 7.1 presents a summary of the published papers that reinforced the importance of our work and show our contributions so far.

Title	Authors	Publisher	Year	Status	Qualis CAPES
Autonomous vehicle steering wheel estimation from a video using multichannel convolutional neural networks	A. E. T. Ferreira; **ALMEIDA, A. P. G. S. de**; VIDAL, F. de B.	International Conference on Informatics in Control, Automation and Robotics (ICINCO)	2018	Published	A4
Document classification using a Bi-LSTM to unclog Brazil's supreme court	Fabricio Ataides Braz, Nilton Correia da Silva, Teofilo Emidio de Campos, Felipe Borges S Chaves, Marcelo HS Ferreira, Pedro Henrique Inazawa, Victor HD Coelho, Bernardo Pablo Sukiennik, **Ana Paula Goncalves Soares de Almeida**, Flavio Barros Vidal, Davi Alves Bezerra, Davi B Gusmao, Gabriel G Ziegler, Ricardo VC Fernandes, Roberta Zumblick, Fabiano Hartmann Peixoto	Neural Information Processing Systems (NeurIPS)	2018	Published	A1
Plant diseases recognition from digital images using multichannel convolutional neural networks	ABADE, A.; **ALMEIDA, A. P. G. S. de**; VIDAL, F. de B.	International Conference on Computer Vision Theory and Applications (VISAPP)	2019	Published	A3
L-CNN: a lattice cross-fusion strategy for multistream convolutional neural networks	**ALMEIDA, A. P. G. de**; VIDAL, F. de B.	Electronics Letters	2019	Published	A4
Sequence-aware multimodal page classification of Brazilian legal documents	Pedro H. Luz de Araujo; **Ana Paula G. S. de Almeida**; Fabricio A. Braz; Nilton C. da Silva; Flavio de Barros Vidal; Teofilo E. de Campos	International Journal on Document Analysis and Recognition (IJDAR)	2022	Accepted for publication	A3
Turning old models fashion again: Recycling classical CNN networks using the Lattice Transformation	**Ana Paula G. S. de Almeida**; Flavio de Barros Vidal	IEEE Access	2022	Under Review	A2

Table 7.1: Publications' summary.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] M. Ma, N. Marturi, Y. Li, A. Leonardis, and R. Stolkin, “Region-sequence based six-stream cnn features for general and fine-grained human action recognition in videos,” *Pattern Recognition*, vol. 76, pp. 506–521, 2018.

[6] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1725–1732.

[7] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in neural information processing systems*, 2014, pp. 568–576.

[8] P. Velickovic, D. Wang, N. D. Lane, and P. Lio, “X-cnn: Cross-modal convolutional neural networks for sparse datasets,” *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, Dec 2016. [Online]. Available: http://dx.doi.org/10.1109/SSCI.2016.7849978

[9] H. R. Vaezi Joze, A. Shaban, M. L. Iuzzolino, and K. Koishida, “Mmtm: Multimodal transfer module for cnn fusion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 286–13 296.

[10] A. Krizhevsky *et al.*, “Learning multiple layers of features from tiny images,” University of Toronto, Tech. Rep., 2009.

[11] A. E. T. Ferreira, A. P. G. S. de Almeida, and F. de Barros Vidal, “Autonomous vehicle steering wheel estimation from a video using multichannel convolutional neural networks,” in *International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2018, pp. 527–534.

[12] A. Abade, A. P. G. S. de Almeida, and F. de Barros Vidal, “Plant diseases recognition from digital images using multichannel convolutional neural networks,” in *14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*, 2019, pp. 450–458.

[13] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[14] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[15] S. Haykin, *Neural networks: a comprehensive foundation.* Prentice Hall PTR, 1994.

[16] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[17] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[18] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[19] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[20] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention.* Springer, 2015, pp. 234–241.

[21] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[23] J. Redmon and A. Farhadi, “Yolo9000: better, faster, stronger,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7263–7271.

[24] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 35, no. 1, pp. 221–231, 2012.

[25] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, “Long-term recurrent convolutional networks for visual recognition and description,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[26] J. Salas, F. de Barros Vidal, and F. Martínez-Trinidad, “Deep learning: Current state,” *IEEE Latin America Transactions*, vol. 17, no. 12, p. 1925–1945, Feb. 2020. [Online]. Available: https://latamt.ieeer9.org/index.php/transactions/article/view/2717

[27] H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, “Two stream lstm: A deep fusion framework for human action recognition,” in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017, pp. 177–186.

[28] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Convolutional two-stream network fusion for video action recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1933–1941.

[29] Q. Xuan, B. Fang, Y. Liu, J. Wang, J. Zhang, Y. Zheng, and G. Bao, “Automatic pearl classification machine based on a multistream convolutional neural network,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 8, pp. 6538–6547, Aug 2018.

[30] D. T. Concha, H. D. A. Maia, H. Pedrini, H. Tacon, A. D. S. Brito, H. D. L. Chaves, and M. B. Vieira, “Multi-stream convolutional neural networks for action recognition in video sequences based on adaptive visual rhythms,” in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 473–480.

[31] Z. Tu, W. Xie, J. Dauwels, B. Li, and J. Yuan, “Semantic cues enhanced multimodality multistream CNN for action recognition,” *IEEE Trans. Circuits Syst. Video Techn.*, vol. 29, no. 5, pp. 1423–1437, 2019. [Online]. Available: https://doi.org/10.1109/TCSVT.2018.2830102

[32] C. Ge, I. Y. Gu, A. S. Jakola, and J. Yang, “Deep learning and multi-sensor fusion for glioma classification using multistream 2d convolutional networks,” in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, July 2018, pp. 5894–5897.

[33] A. Monteiro, M. de Oliveira, R. de Oliveira, and T. da Silva, “Embedded application of convolutional neural networks on raspberry pi for shm,” *Electronics Letters*, vol. 54, no. 11, pp. 680–682, 2018.

[34] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[35] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[36] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[37] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 91–99. [Online]. Available: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[38] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, “Large-scale long-tailed recognition in an open world,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2537–2546.

[39] H.-Z. Wang, G.-Q. Li, G.-B. Wang, J.-C. Peng, H. Jiang, and Y.-T. Liu, “Deep Learning based Ensemble approach for Probabilistic Wind Power Forecasting,” *Applied Energy*, vol. 188, pp. 56 – 70, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306261916317421

[40] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[41] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[42] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, “Phoneme recognition using time-delay neural networks,” *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[43] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems*, 1990, pp. 396–404.

[44] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[46] A. for Computer Machinery, “Acm digital library website,” feb 2022. [Online]. Available: https://dl.acm.org/

[47] I. of Electrical and E. E. (IEEE), “Ieee xplore website,” feb 2022. [Online]. Available: https://ieeexplore.ieee.org/

[48] E. B.V., “Science direct website,” feb 2022. [Online]. Available: https://www.sciencedirect.com/

[49] Y. Zhou, J. Deng, I. Kotsia, and S. Zafeiriou, “Dense 3d face decoding over 2500fps: Joint texture & shape convolutional mesh decoders,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1097–1106.

[50] R. D. Singh, A. Mittal, and R. Bhatia, “3D Convolutional Neural Network for Object Recognition: A Review,” *Multimedia Tools and Applications*, vol. 78, no. 12, pp. 15 951–15 995, 2019. [Online]. Available: https://doi.org/10.1007/s11042-018-6912-6

[51] H. Lei, N. Akhtar, and A. Mian, "Octree guided cnn with spherical kernels for 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9631–9640.

[52] B. Vandersmissen, N. Knudde, A. Jalalvand, I. Couckuyt, T. Dhaene, and W. De Neve, "Indoor Human Activity Recognition using High-dimensional Sensors and Deep Neural nNetworks," *Neural Computing and Applications*, Aug 2019. [Online]. Available: https://doi.org/10.1007/s00521-019-04408-1

[53] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1251–1258.

[54] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4700–4708.

[55] L. Bottou, F. F. Soulie, P. Blanchet, and J.-S. Liénard, "Speaker-independent isolated digit recognition: Multilayer perceptrons vs. dynamic time warping," *Neural Networks*, vol. 3, no. 4, pp. 453–465, 1990.

[56] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 8, no. 1, pp. 98–113, 1997.

[57] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 35, no. 1, pp. 221–231, 2012.

[58] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[59] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[60] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[61] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[62] R. Girshick, "Fast r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015, pp. 1440–1448.

[63] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[64] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2414–2423.

[65] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 525–542.

[66] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 2961–2969.

[67] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[68] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie, "Behavior recognition via sparse spatio-temporal features," in *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*. IEEE, 2005, pp. 65–72.

[69] I. Laptev, "On space-time interest points," *International Journal of Computer Vision (IJCV)*, vol. 64, no. 2-3, pp. 107–123, 2005.

[70] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2008, pp. 1–8.

[71] J. Liu, Jiebo Luo, and M. Shah, "Recognizing realistic actions from videos "in the wild"," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2009, pp. 1996–2003.

[72] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, “Evaluation of local spatio-temporal features for action recognition,” in *British Machine Vision Conference (BMVC).* BMVA Press, 2009, pp. 124–1.

[73] J. C. Niebles, C.-W. Chen, and L. Fei-Fei, “Modeling temporal structure of decomposable motion segments for activity classification,” in *European Conference on Computer Vision (ECCV).* Springer, 2010, pp. 392–405.

[74] H. Wang, A. Kläser, C. Schmid, and C.-L. Liu, “Action recognition by dense trajectories,” in *CVPR 2011.* IEEE, 2011, pp. 3169–3176.

[75] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, “Deep neural networks segment neuronal membranes in electron microscopy images,” in *Advances in neural information processing systems*, 2012, pp. 2843–2851.

[76] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 35, no. 8, pp. 1915–1929, 2012.

[77] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “Cnn features off-the-shelf: an astounding baseline for recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 806–813.

[78] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *arXiv preprint arXiv:1312.6229*, 2013.

[79] Z. Wu, Y.-G. Jiang, X. Wang, H. Ye, X. Xue, and J. Wang, “Fusing multi-stream deep networks for video classification,” *arXiv preprint arXiv:1509.06086*, 2015.

[80] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao, “Towards good practices for very deep two-stream convnets,” *arXiv preprint arXiv:1507.02159*, 2015.

[81] J. Yang, Y. Zhao, J. C.-W. Chan, and C. Yi, “Hyperspectral image classification using two-channel deep convolutional neural network,” in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS).* IEEE, 2016, pp. 5079–5082.

[82] X. Peng and C. Schmid, “Multi-region two-stream r-cnn for action detection,” in *European Conference on Computer Vision (ECCV).* Springer, 2016, pp. 744–759.

[83] A. Diba, A. M. Pazandeh, and L. Van Gool, “Efficient two-stream motion and appearance 3d cnns for video classification,” *arXiv preprint arXiv:1608.08851*, 2016.

[84] B. Singh, T. K. Marks, M. Jones, O. Tuzel, and M. Shao, "A multi-stream bi-directional recurrent neural network for fine-grained action detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1961–1970.

[85] D. Chung, K. Tahboub, and E. J. Delp, "A two stream siamese convolutional neural network for person re-identification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1983–1991.

[86] T. Akilan, Q. J. Wu, A. Safaei, and W. Jiang, "A late fusion approach for harnessing multi-cnn model high-level features," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 566–571.

[87] Y. Yu and F. Liu, "A two-stream deep fusion framework for high-resolution aerial scene classification," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[88] P. Veličković, L. Karazija, N. D. Lane, S. Bhattacharya, E. Liberis, P. Liò, A. Chieh, O. Bellahsen, and M. Vegreville, "Cross-modal recurrent models for weight objective prediction from multimodal time-series data," *Proceedings of the 12th EAI International Conference on Pervasive Computing Technologies for Healthcare*, May 2018. [Online]. Available: http://dx.doi.org/10.1145/3240925.3240937

[89] M. Amin-Naji, A. Aghagolzadeh, and M. Ezoji, "Ensemble of cnn for multi-focus image fusion," *Information Fusion*, vol. 51, pp. 201–214, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253518306043

[90] J.-M. Pérez-Rúa, V. Vielzeuf, S. Pateux, M. Baccouche, and F. Jurie, "Mfas: Multimodal fusion architecture search," pp. 6966–6975, 2019.

[91] A. P. G. S. de Almeida and F. de Barros Vidal, "L-cnn: a lattice cross-fusion strategy for multistream convolutional neural networks," *Electronics Letters*, vol. 55, pp. 1180–1182(2), October 2019. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/el.2019.2631

[92] C. Cangea, P. Veličković, and P. Liò, "Xflow: Cross-modal deep neural networks for audiovisual classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3711–3720, 2020.

[93] B. P. Lathi, *Linear Systems and Signals*, 2nd ed. USA: Oxford University Press, Inc., 2009.

[94] F. Chollet *et al.*, “Keras,” https://github.com/fchollet/keras, 2015.

[95] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[96] Y. LeCun, F. J. Huang, and L. Bottou, “Learning methods for generic object recognition with invariance to pose and lighting,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR’04. USA: IEEE Computer Society, 2004, p. 97–104.

[97] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, “Bdd100k: A diverse driving dataset for heterogeneous multitask learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020, pp. 2636–2645.

[98] D. Hughes, M. Salathé *et al.*, “An open access repository of images on plant health to enable the development of mobile disease diagnostics,” *arXiv preprint arXiv:1511.08060*, 2015.

[99] S. Mohanty, D. Hughes, and M. Salathé, “Using deep learning for image-based plant disease detection,” *Frontiers in Plant Science*, vol. 7, no. September, 2016.

[100] P. H. Luz de Araujo, T. E. de Campos, F. Ataides Braz, and N. Correia da Silva, “VICTOR: a dataset for Brazilian legal documents classification,” in *Proceedings of the 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 1449–1458. [Online]. Available: https://aclanthology.org/2020.lrec-1.181

[101] D. M. Powers, “Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation,” 2011.

[102] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313. [Online]. Available: http://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf

[103] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. [Online]. Available: http://dx.doi.org/10.1002/rob.20147

[104] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 10, no. 3, pp. 362 – 373, May 1988.

[105] Q. Wang, J. Gao, and Y. Yuan, "Embedding structured contour and location prior in siamesed fully convolutional networks for road detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 230–241, Jan 2018.

[106] T. Rumpf, A.-K. Mahlein, U. Steiner, E.-C. Oerke, H.-W. Dehne, and L. Plümer, "Early detection and classification of plant diseases with support vector machines based on hyperspectral reflectance," *Computers and Electronics in Agriculture*, vol. 74, no. 1, pp. 91 – 99, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169910001262

[107] H. Al-hiary, S. Bani-ahmad, M. Reyalat, M. Braik, and Z. Alrahamneh, "Fast and accurate detection and classification of plant diseases," *International Journal of Computer Applications*, vol. 17, no. 1, pp. 31 – 38, march 2011.

[108] S. B. Patil and S. K. Bodhe, "Leaf disease severity measurement using image processing," *International Journal of Engineering and Technology*, vol. 3, no. 5, pp. 297–301, 2011.

[109] V. Singh and A. Misra, "Detection of plant leaf diseases using image segmentation and soft computing techniques," *Information Processing in Agriculture*, vol. 4, no. 1, pp. 41 – 49, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214317316300154

[110] R. Pydipati, T. Burks, and W. Lee, "Identification of citrus disease using color texture features and discriminant analysis," *Computers and Electronics in Agriculture*, vol. 52, no. 1, pp. 49 – 59, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168169906000287

[111] M. F. A. Jabal, S. Hamid, S. Shuib, I. Ahmad, M. F. A. Jabal, S. Hamid, S. Shuib, and I. Ahmad, "Leaf Features Extraction and Recognition Approaches To Classify Plant," *Journal of Computer Science*, vol. 9, no. 10, pp. 1295–1304, oct 2013. [Online]. Available: http://thescipub.com/abstract/10.3844/jcssp.2013.1295.1304

[112] K. P. Ferentinos, "Deep learning models for plant disease detection and diagnosis," *Computers and Electronics in Agriculture*, vol. 145, pp. 311 – 318, 2018.

[113] F. A. Braz, N. C. da Silva, T. E. de Campos, F. B. S. Chaves, M. H. Ferreira, P. H. Inazawa, V. H. Coelho, B. P. Sukiennik, A. P. G. S. de Almeida, F. B. Vidal *et al.*, "Document classification using a bi-lstm to unclog brazil's supreme court," *arXiv preprint arXiv:1811.11569*, 2018.

[114] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[115] J. Howard and S. Gugger, "fastai: A layered API for deep learning," *CoRR*, vol. abs/2002.04688, 2020. [Online]. Available: https://arxiv.org/abs/2002.04688

[116] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[117] Y. Xu, Y. Xu, T. Lv, L. Cui, F. Wei, G. Wang, Y. Lu, D. A. F. Florêncio, C. Zhang, W. Che, M. Zhang, and L. Zhou, "Layoutlmv2: Multi-modal pre-training for visually-rich document understanding," *CoRR*, vol. abs/2012.14740, 2020. [Online]. Available: https://arxiv.org/abs/2012.14740

[118] Z. Tu, W. Xie, Q. Qin, R. Poppe, R. Veltkamp, B. Li, and J. Yuan, "Multi-stream cnn: Learning representations based on human-related regions for action recognition," *Pattern Recognition*, vol. 79, pp. 32–43, 7 2018.

[119] S. M. Azar, M. G. Atigh, and A. Nickabadi, "A multi-stream convolutional neural network framework for group activity recognition," *CoRR*, vol. abs/1812.10328, 2018. [Online]. Available: http://arxiv.org/abs/1812.10328

[120] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and Systems (2nd Ed.)*. USA: Prentice-Hall, Inc., 1996.

[121] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.

[122] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-100 (canadian institute for advanced research),” University of Toronto, Tech. Rep., 2009. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[123] P. Bhargava, “On generalizing detection models for unconstrained environments,” *CoRR*, vol. abs/1909.13080, 2019. [Online]. Available: http://arxiv.org/abs/1909.13080

Appendix

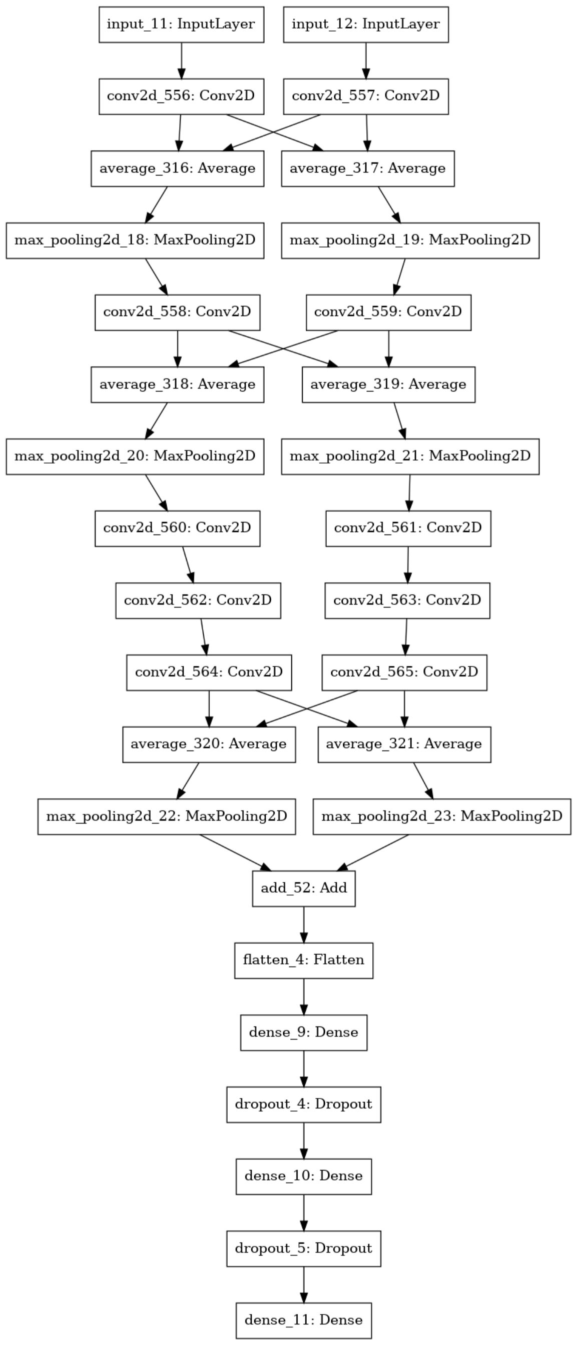
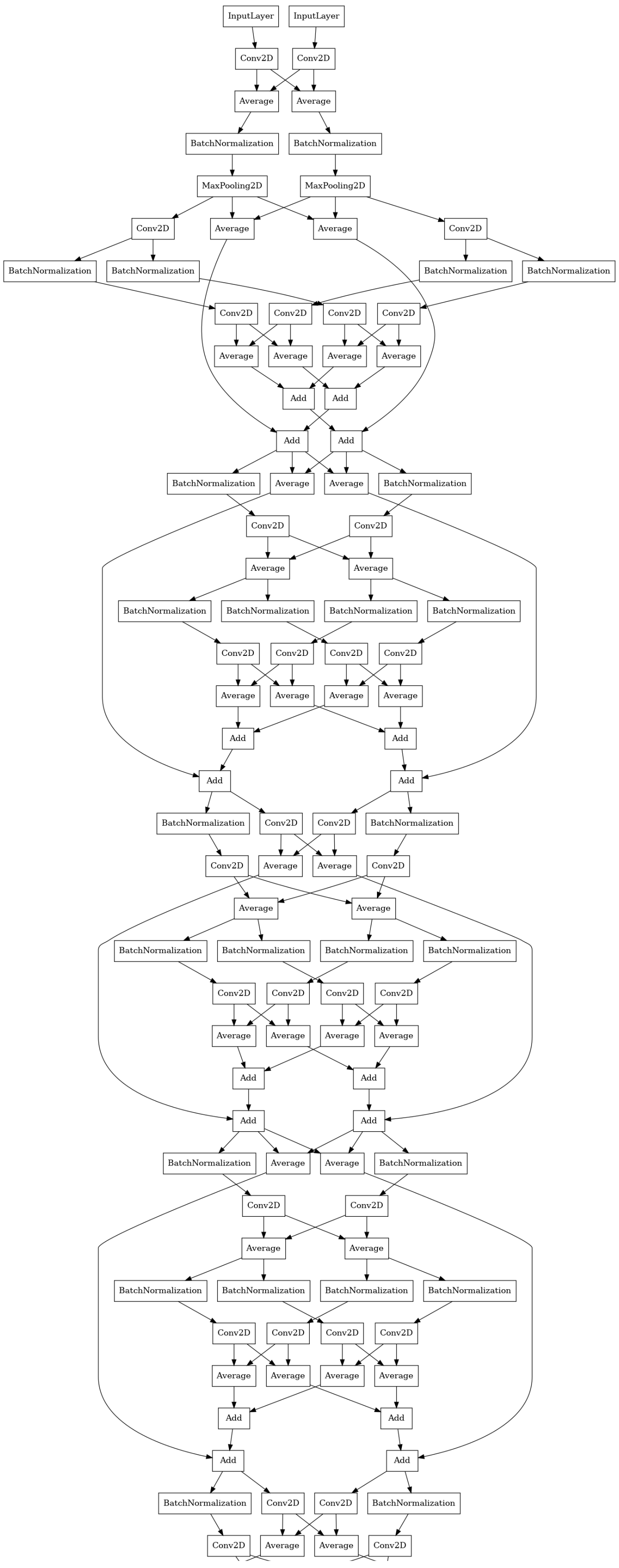


Figure 1: L-Alexnet.


InputLayer
InputLayer
Conv2D
Average
BatchNormalization
MaxPooling2D
Add

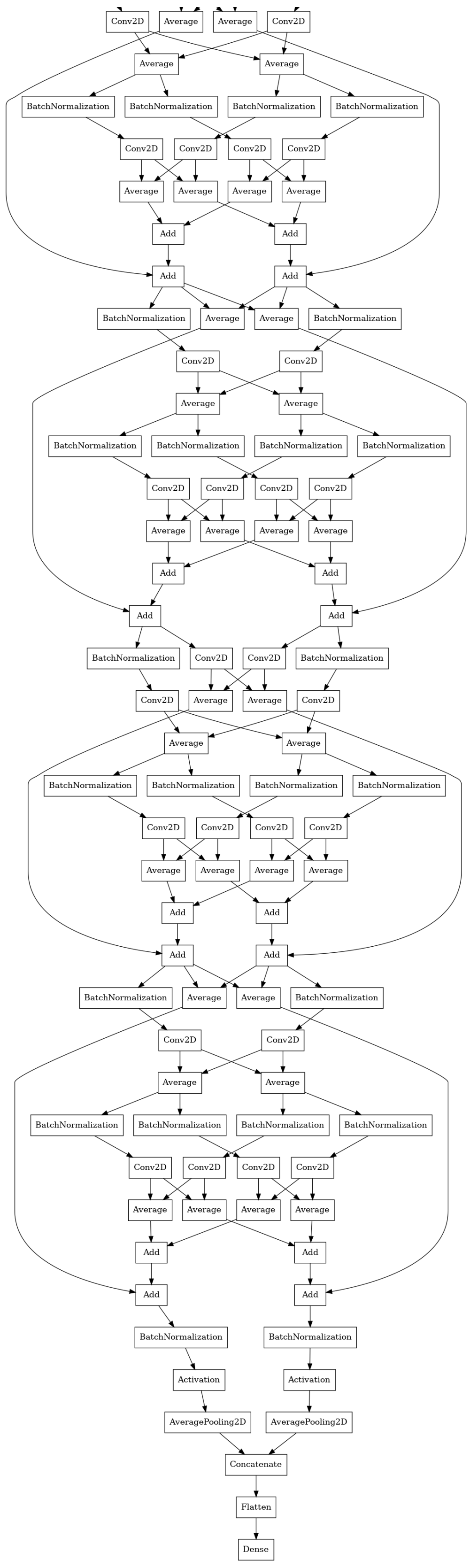
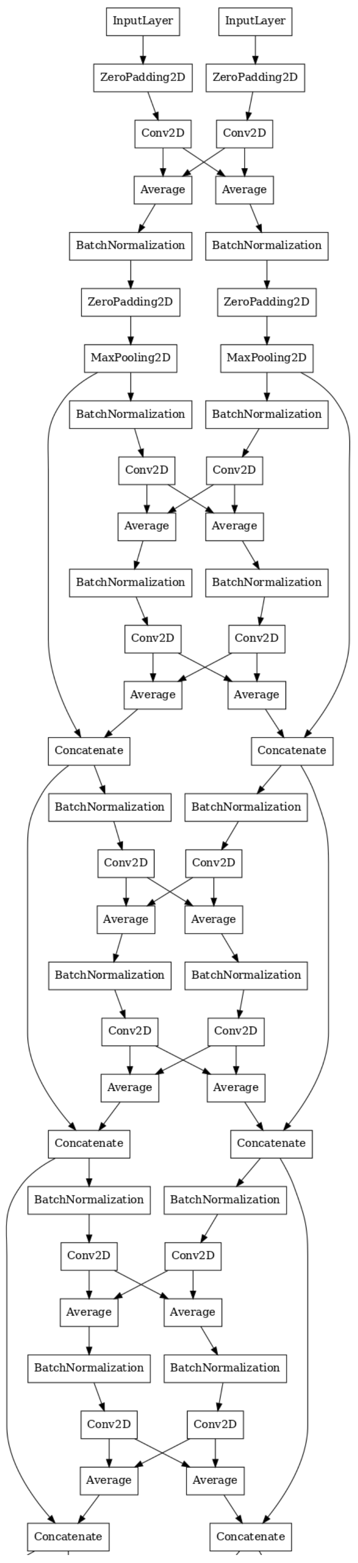
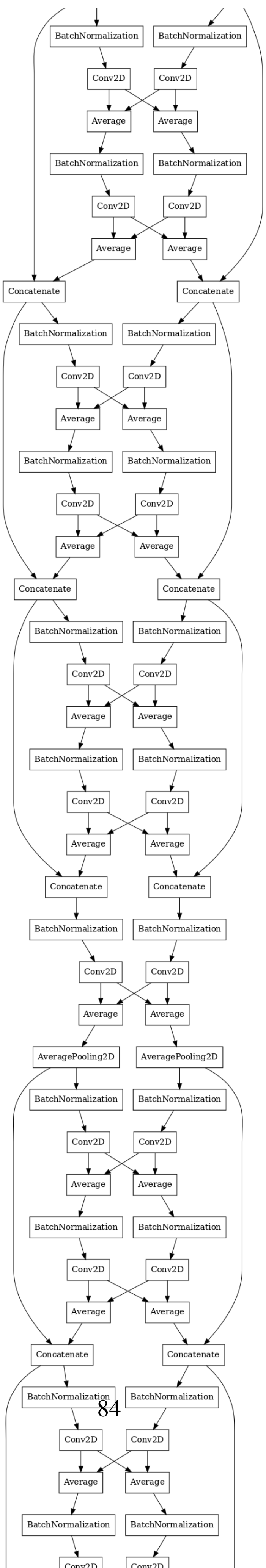
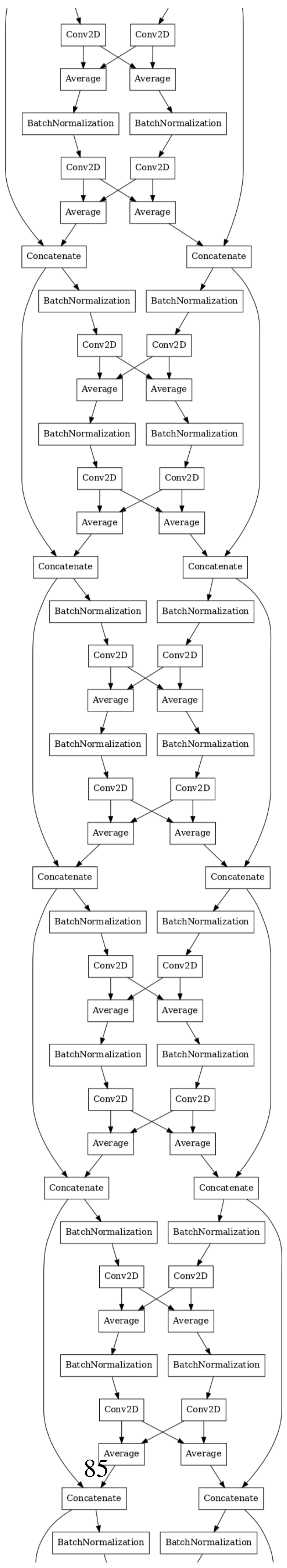


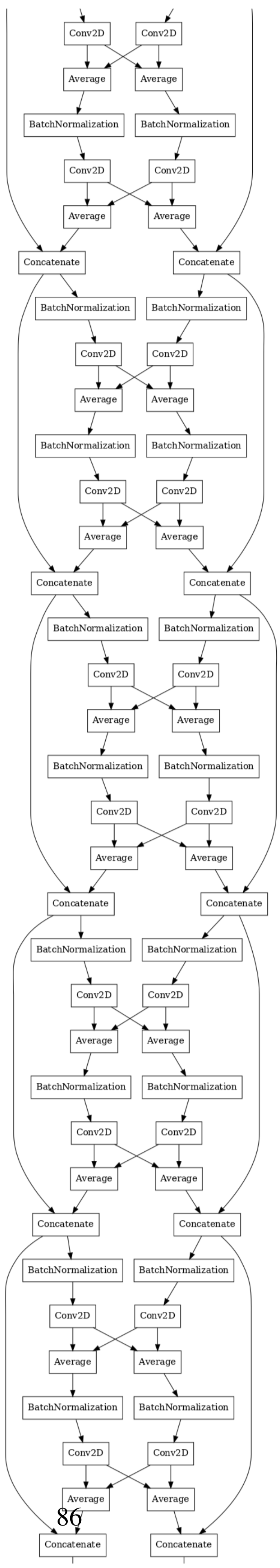
Figure 2: L-ResNet-18.

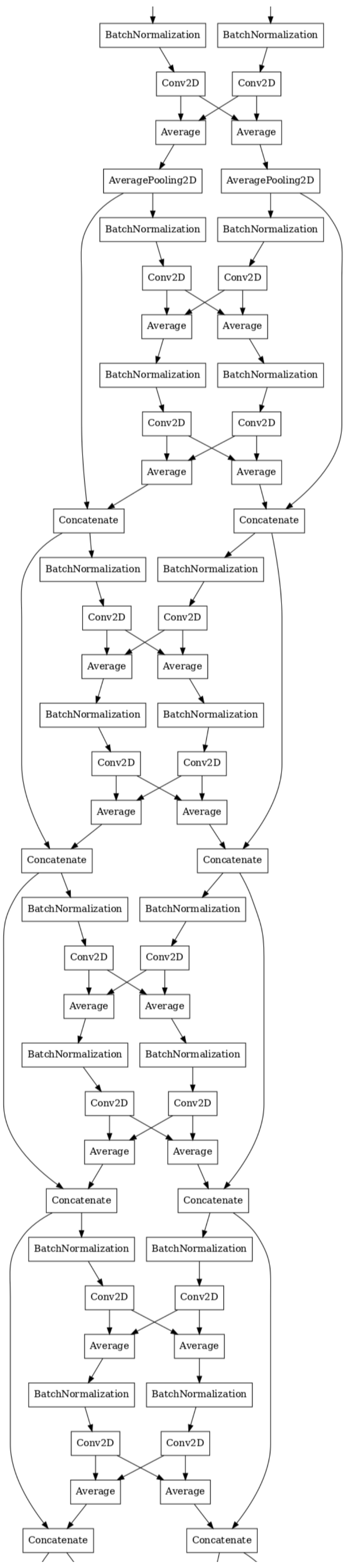

InputLayer
InputLayer
ZeroPadding2D
ZeroPadding2D
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
ZeroPadding2D
ZeroPadding2D
MaxPooling2D
MaxPooling2D
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate

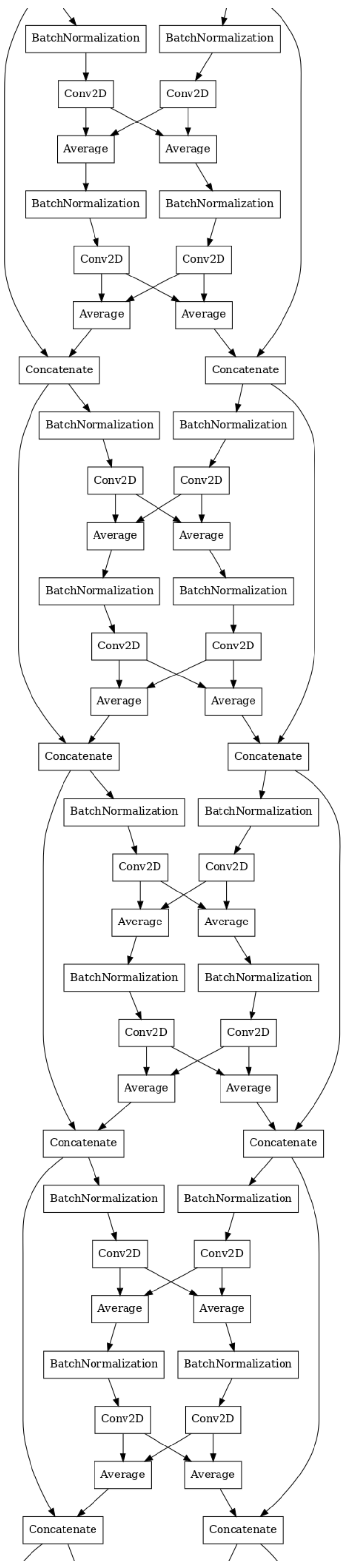

BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
AveragePooling2D
AveragePooling2D
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D

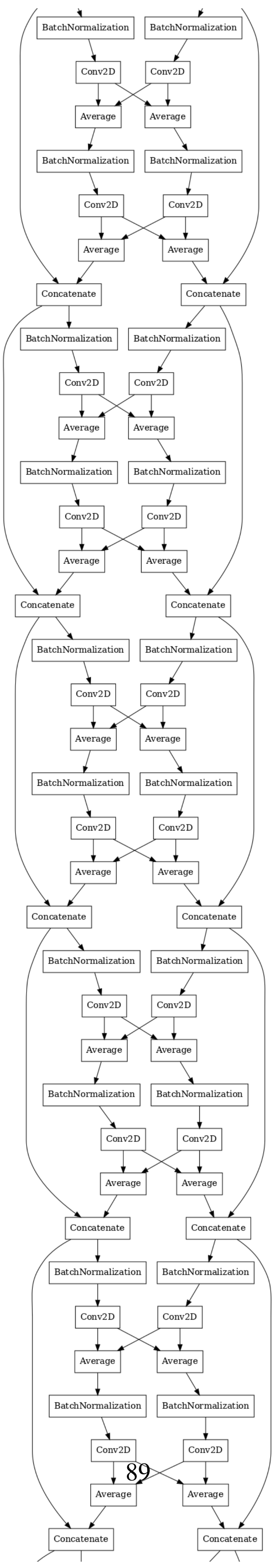


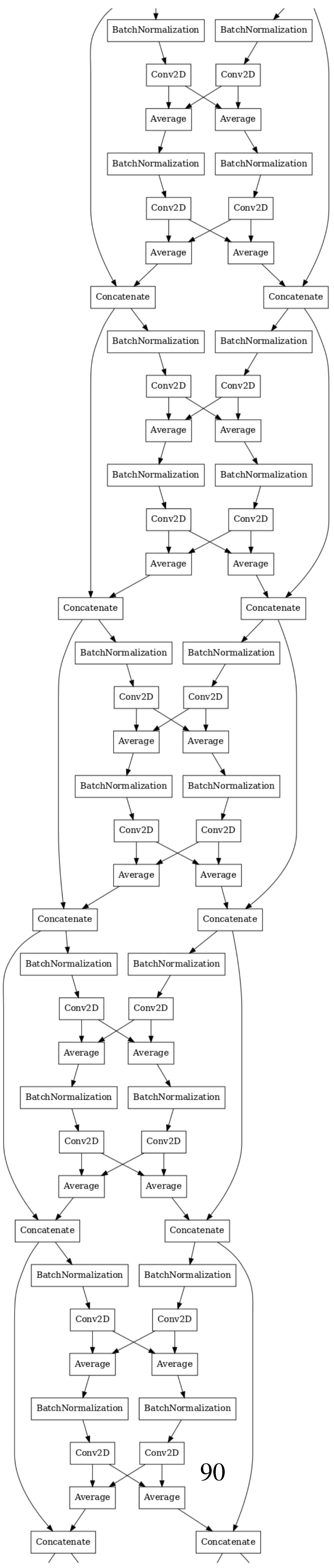
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization

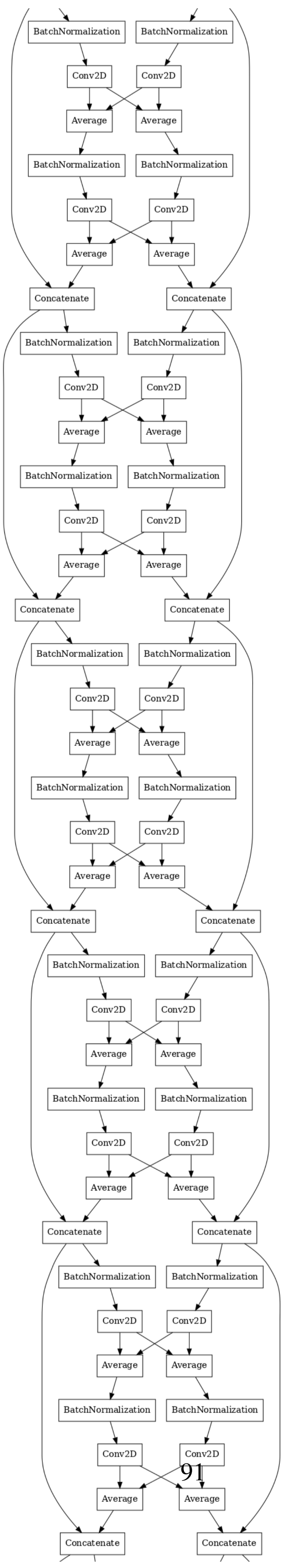

Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
Conv2D
Average
Concatenate

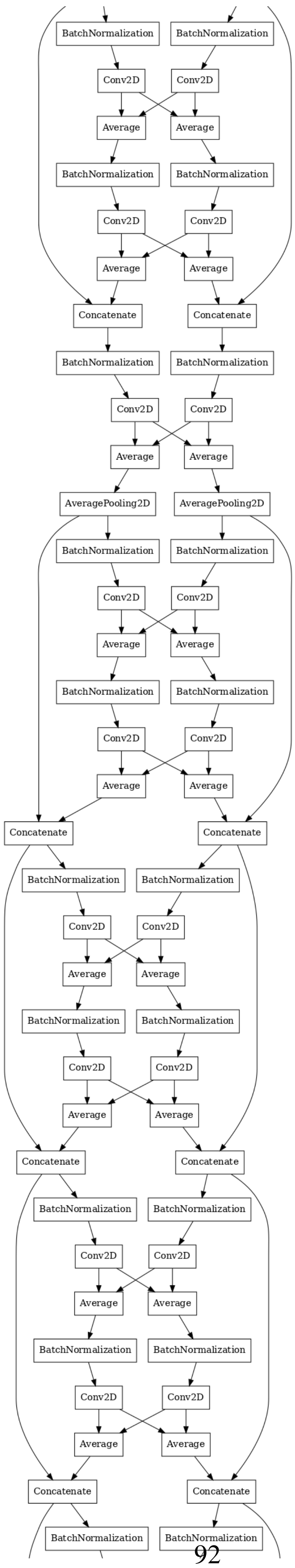

BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
AveragePooling2D
AveragePooling2D
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate

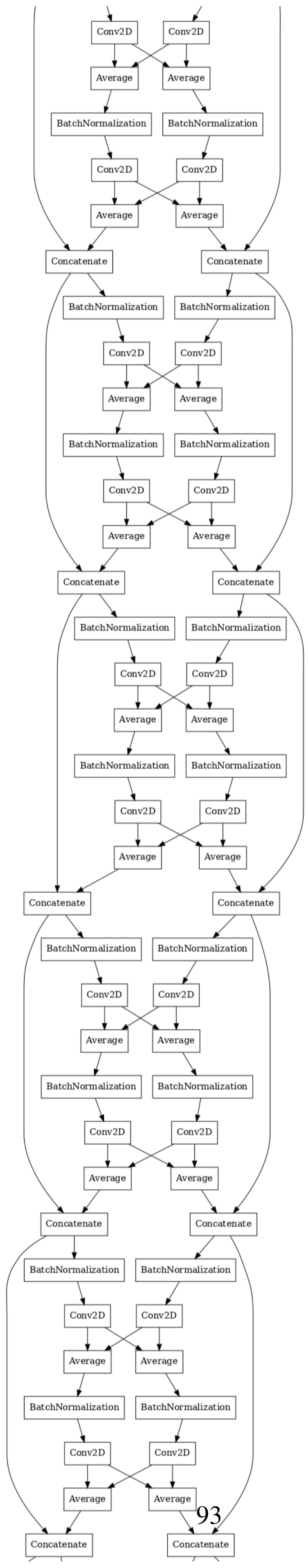

BatchNormalization
Conv2D
Average
Concatenate

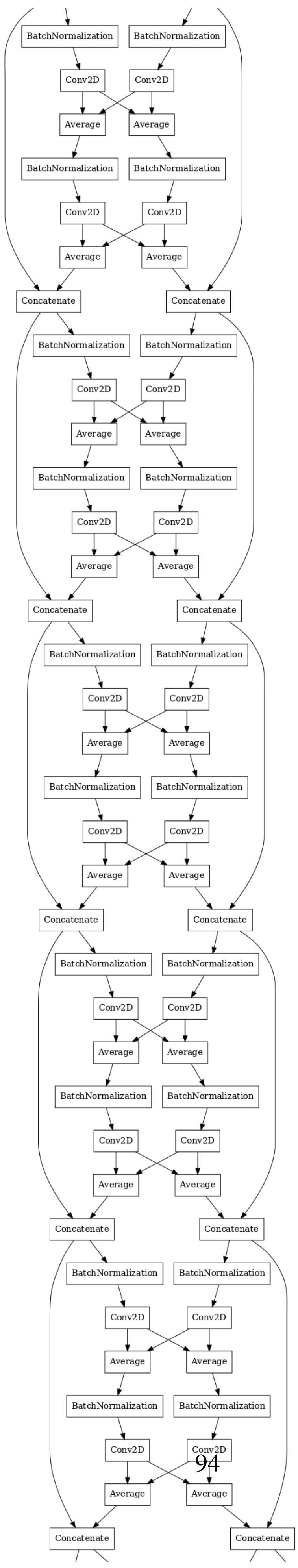

BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
BatchNormalization
BatchNormalization
Conv2D
Conv2D
Average
Average
Concatenate
Concatenate


BatchNormalization
Conv2D
Average
Concatenate


BatchNormalization
Conv2D
Average
Concatenate


BatchNormalization
Conv2D
Average
Concatenate
AveragePooling2D


Conv2D
Average
BatchNormalization
Concatenate


BatchNormalization
Conv2D
Average
Concatenate

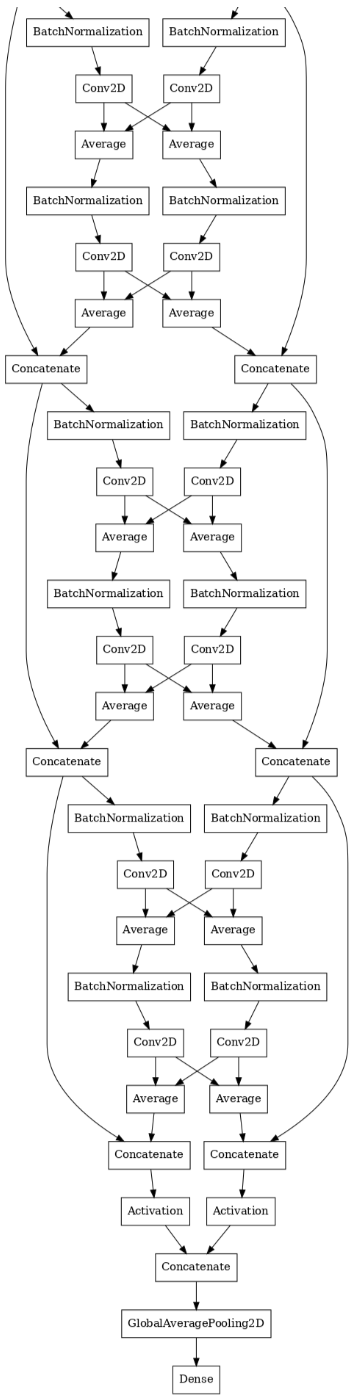
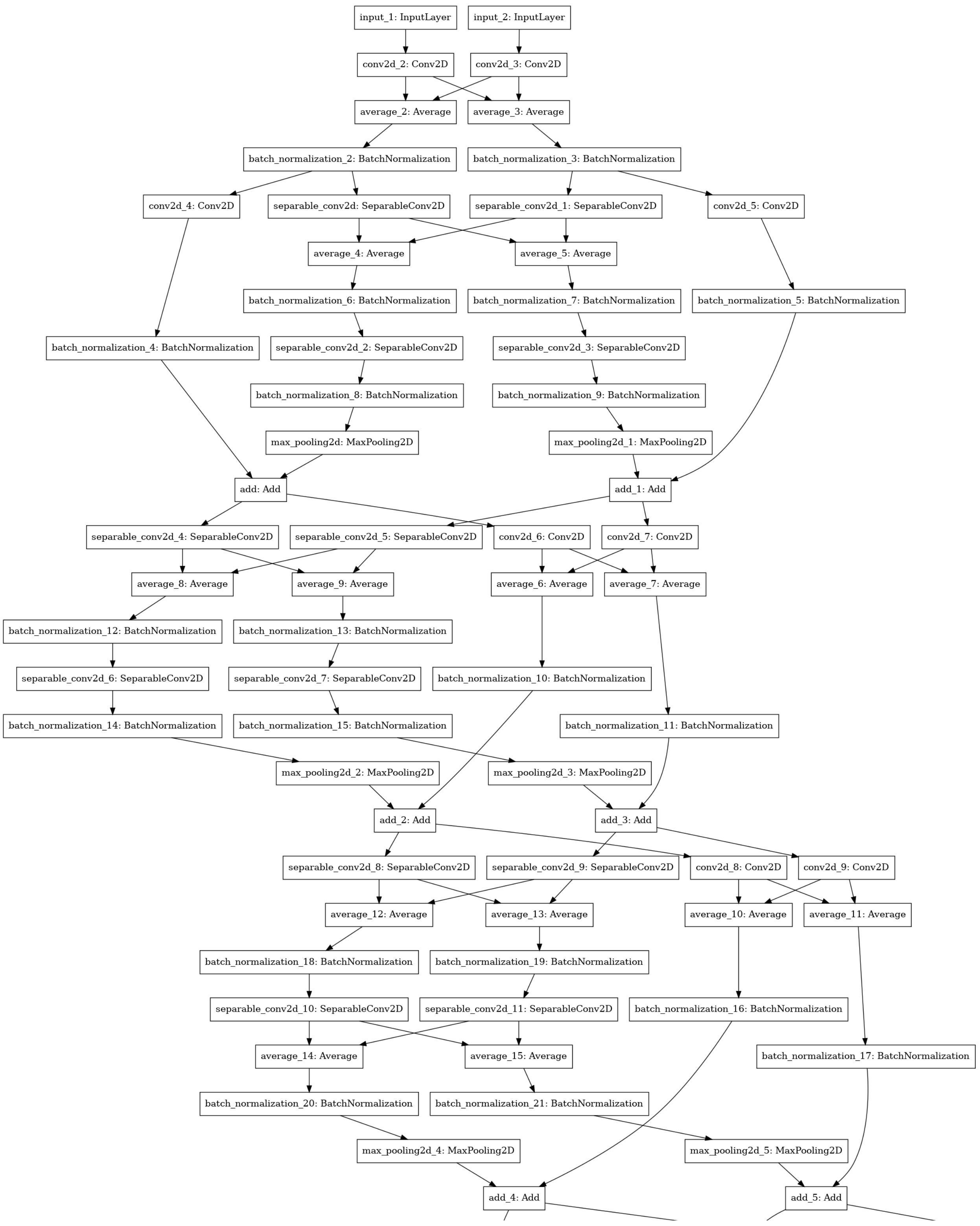
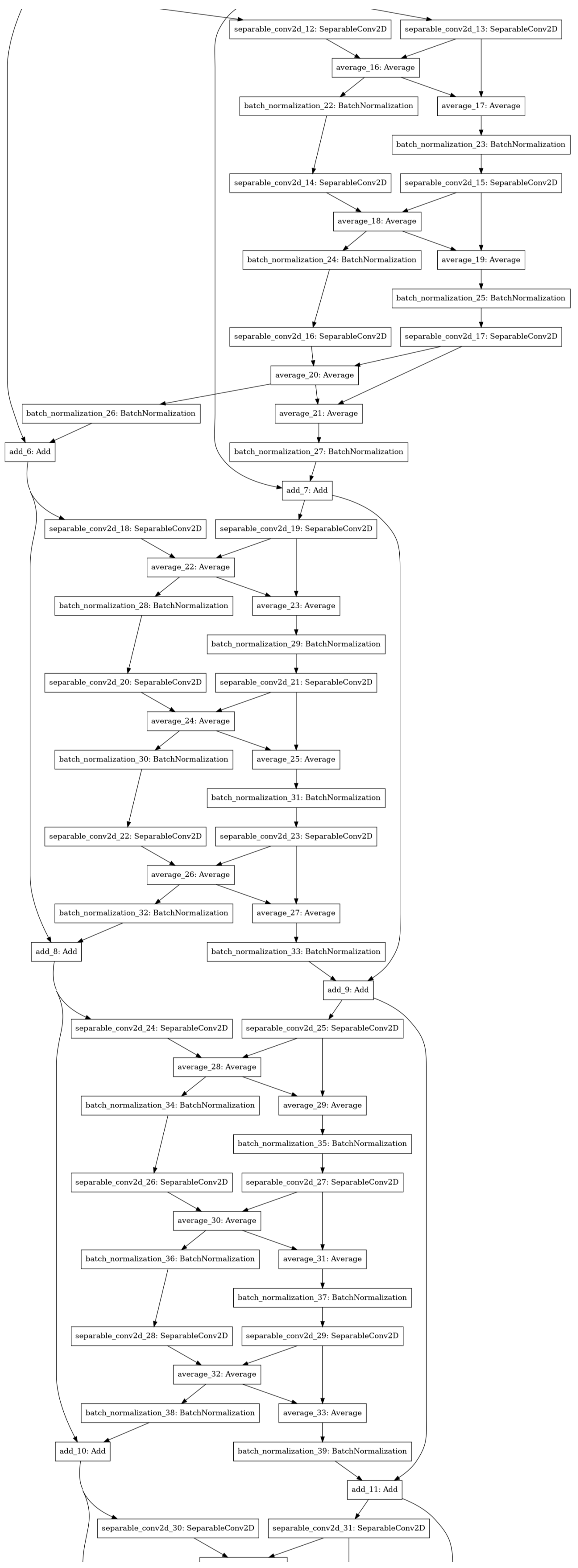
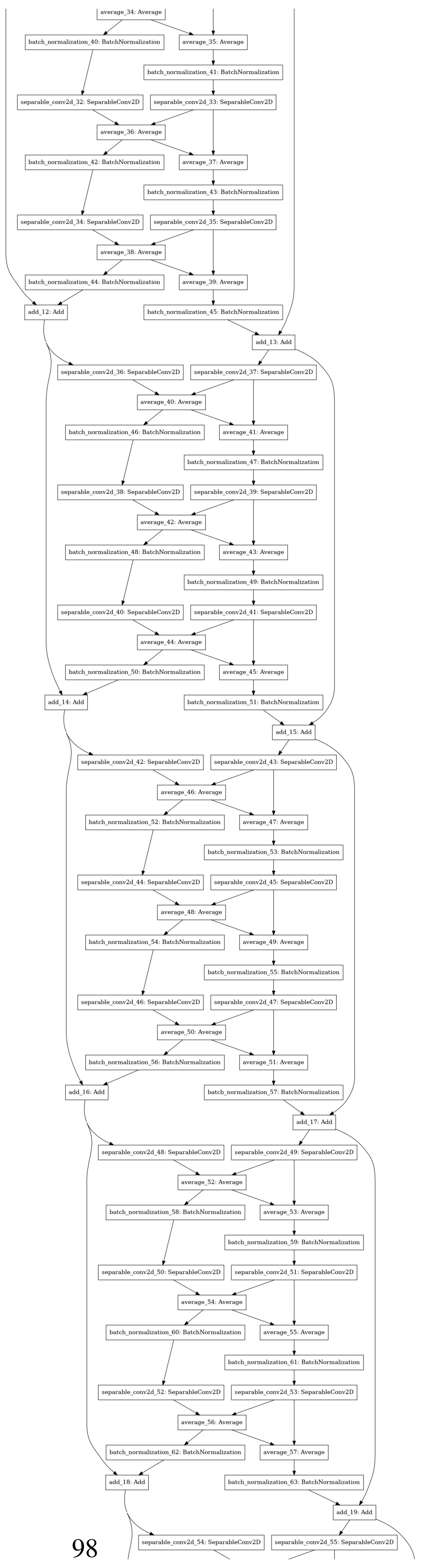


Figure 3: L-DenseNet-121.


input_1: InputLayer
input_2: InputLayer
conv2d_2: Conv2D
conv2d_3: Conv2D
average_2: Average
average_3: Average
batch_normalization_2: BatchNormalization
batch_normalization_3: BatchNormalization
conv2d_4: Conv2D
separable_conv2d: SeparableConv2D
separable_conv2d_1: SeparableConv2D
conv2d_5: Conv2D
average_4: Average
average_5: Average
batch_normalization_6: BatchNormalization
batch_normalization_7: BatchNormalization
batch_normalization_5: BatchNormalization
batch_normalization_4: BatchNormalization
separable_conv2d_2: SeparableConv2D
separable_conv2d_3: SeparableConv2D
batch_normalization_8: BatchNormalization
batch_normalization_9: BatchNormalization
max_pooling2d: MaxPooling2D
max_pooling2d_1: MaxPooling2D
add: Add
add_1: Add
separable_conv2d_4: SeparableConv2D
separable_conv2d_5: SeparableConv2D
conv2d_6: Conv2D
conv2d_7: Conv2D
average_8: Average
average_9: Average
average_6: Average
average_7: Average
batch_normalization_12: BatchNormalization
batch_normalization_13: BatchNormalization
separable_conv2d_6: SeparableConv2D
separable_conv2d_7: SeparableConv2D
batch_normalization_10: BatchNormalization
batch_normalization_14: BatchNormalization
batch_normalization_15: BatchNormalization
batch_normalization_11: BatchNormalization
max_pooling2d_2: MaxPooling2D
max_pooling2d_3: MaxPooling2D
add_2: Add
add_3: Add
separable_conv2d_8: SeparableConv2D
separable_conv2d_9: SeparableConv2D
conv2d_8: Conv2D
conv2d_9: Conv2D
average_12: Average
average_13: Average
average_10: Average
average_11: Average
batch_normalization_18: BatchNormalization
batch_normalization_19: BatchNormalization
separable_conv2d_10: SeparableConv2D
separable_conv2d_11: SeparableConv2D
batch_normalization_16: BatchNormalization
batch_normalization_17: BatchNormalization
average_14: Average
average_15: Average
batch_normalization_20: BatchNormalization
batch_normalization_21: BatchNormalization
max_pooling2d_4: MaxPooling2D
max_pooling2d_5: MaxPooling2D
add_4: Add
add_5: Add


separable_conv2d_12: SeparableConv2D
separable_conv2d_13: SeparableConv2D
average_16: Average
batch_normalization_22: BatchNormalization
average_17: Average
batch_normalization_23: BatchNormalization
separable_conv2d_14: SeparableConv2D
separable_conv2d_15: SeparableConv2D
average_18: Average
batch_normalization_24: BatchNormalization
average_19: Average
batch_normalization_25: BatchNormalization
separable_conv2d_16: SeparableConv2D
separable_conv2d_17: SeparableConv2D
average_20: Average
batch_normalization_26: BatchNormalization
average_21: Average
add_6: Add
batch_normalization_27: BatchNormalization
add_7: Add
separable_conv2d_18: SeparableConv2D
separable_conv2d_19: SeparableConv2D
average_22: Average
batch_normalization_28: BatchNormalization
average_23: Average
batch_normalization_29: BatchNormalization
separable_conv2d_20: SeparableConv2D
separable_conv2d_21: SeparableConv2D
average_24: Average
batch_normalization_30: BatchNormalization
average_25: Average
batch_normalization_31: BatchNormalization
separable_conv2d_22: SeparableConv2D
separable_conv2d_23: SeparableConv2D
average_26: Average
batch_normalization_32: BatchNormalization
average_27: Average
add_8: Add
batch_normalization_33: BatchNormalization
add_9: Add
separable_conv2d_24: SeparableConv2D
separable_conv2d_25: SeparableConv2D
average_28: Average
batch_normalization_34: BatchNormalization
average_29: Average
batch_normalization_35: BatchNormalization
separable_conv2d_26: SeparableConv2D
separable_conv2d_27: SeparableConv2D
average_30: Average
batch_normalization_36: BatchNormalization
average_31: Average
batch_normalization_37: BatchNormalization
separable_conv2d_28: SeparableConv2D
separable_conv2d_29: SeparableConv2D
average_32: Average
batch_normalization_38: BatchNormalization
average_33: Average
add_10: Add
batch_normalization_39: BatchNormalization
add_11: Add
separable_conv2d_30: SeparableConv2D
separable_conv2d_31: SeparableConv2D


average_34: Average
batch_normalization_40: BatchNormalization
average_35: Average
batch_normalization_41: BatchNormalization
separable_conv2d_32: SeparableConv2D
separable_conv2d_33: SeparableConv2D
average_36: Average
batch_normalization_42: BatchNormalization
average_37: Average
batch_normalization_43: BatchNormalization
separable_conv2d_34: SeparableConv2D
separable_conv2d_35: SeparableConv2D
average_38: Average
batch_normalization_44: BatchNormalization
average_39: Average
add_12: Add
batch_normalization_45: BatchNormalization
add_13: Add
separable_conv2d_36: SeparableConv2D
separable_conv2d_37: SeparableConv2D
average_40: Average
batch_normalization_46: BatchNormalization
average_41: Average
batch_normalization_47: BatchNormalization
separable_conv2d_38: SeparableConv2D
separable_conv2d_39: SeparableConv2D
average_42: Average
batch_normalization_48: BatchNormalization
average_43: Average
batch_normalization_49: BatchNormalization
separable_conv2d_40: SeparableConv2D
separable_conv2d_41: SeparableConv2D
average_44: Average
batch_normalization_50: BatchNormalization
average_45: Average
add_14: Add
batch_normalization_51: BatchNormalization
add_15: Add
separable_conv2d_42: SeparableConv2D
separable_conv2d_43: SeparableConv2D
average_46: Average
batch_normalization_52: BatchNormalization
average_47: Average
batch_normalization_53: BatchNormalization
separable_conv2d_44: SeparableConv2D
separable_conv2d_45: SeparableConv2D
average_48: Average
batch_normalization_54: BatchNormalization
average_49: Average
batch_normalization_55: BatchNormalization
separable_conv2d_46: SeparableConv2D
separable_conv2d_47: SeparableConv2D
average_50: Average
batch_normalization_56: BatchNormalization
average_51: Average
add_16: Add
batch_normalization_57: BatchNormalization
add_17: Add
separable_conv2d_48: SeparableConv2D
separable_conv2d_49: SeparableConv2D
average_52: Average
batch_normalization_58: BatchNormalization
average_53: Average
batch_normalization_59: BatchNormalization
separable_conv2d_50: SeparableConv2D
separable_conv2d_51: SeparableConv2D
average_54: Average
batch_normalization_60: BatchNormalization
average_55: Average
batch_normalization_61: BatchNormalization
separable_conv2d_52: SeparableConv2D
separable_conv2d_53: SeparableConv2D
average_56: Average
batch_normalization_62: BatchNormalization
average_57: Average
add_18: Add
batch_normalization_63: BatchNormalization
add_19: Add
separable_conv2d_54: SeparableConv2D
separable_conv2d_55: SeparableConv2D

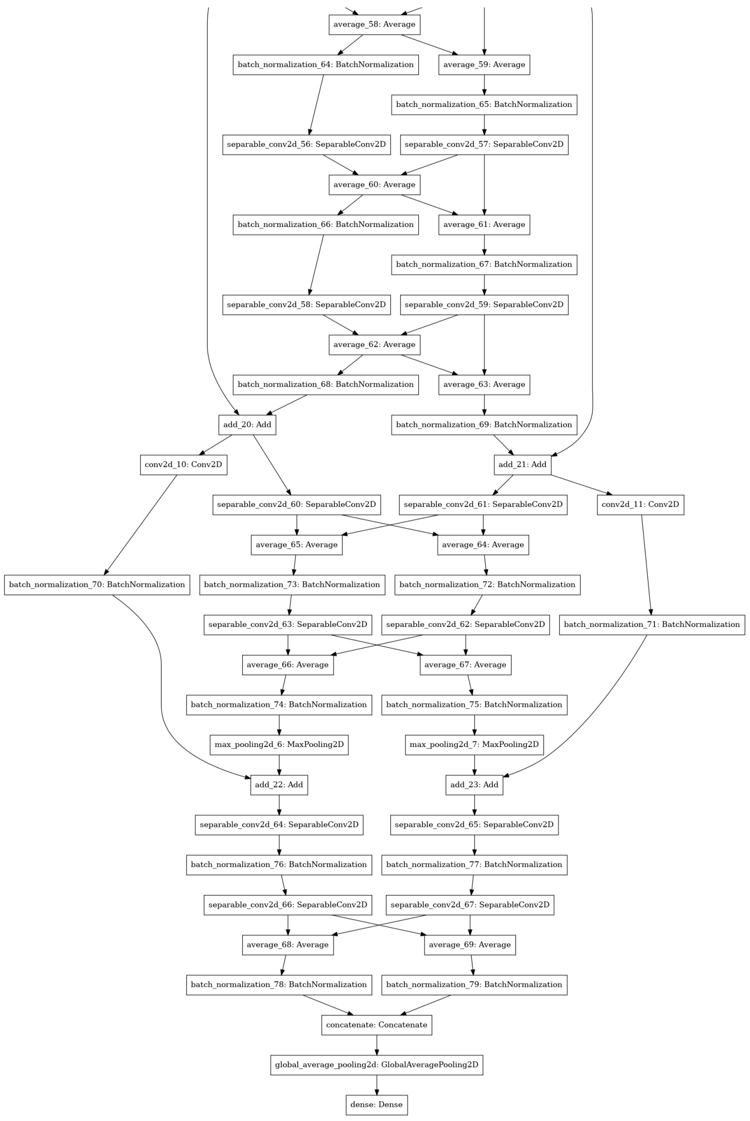
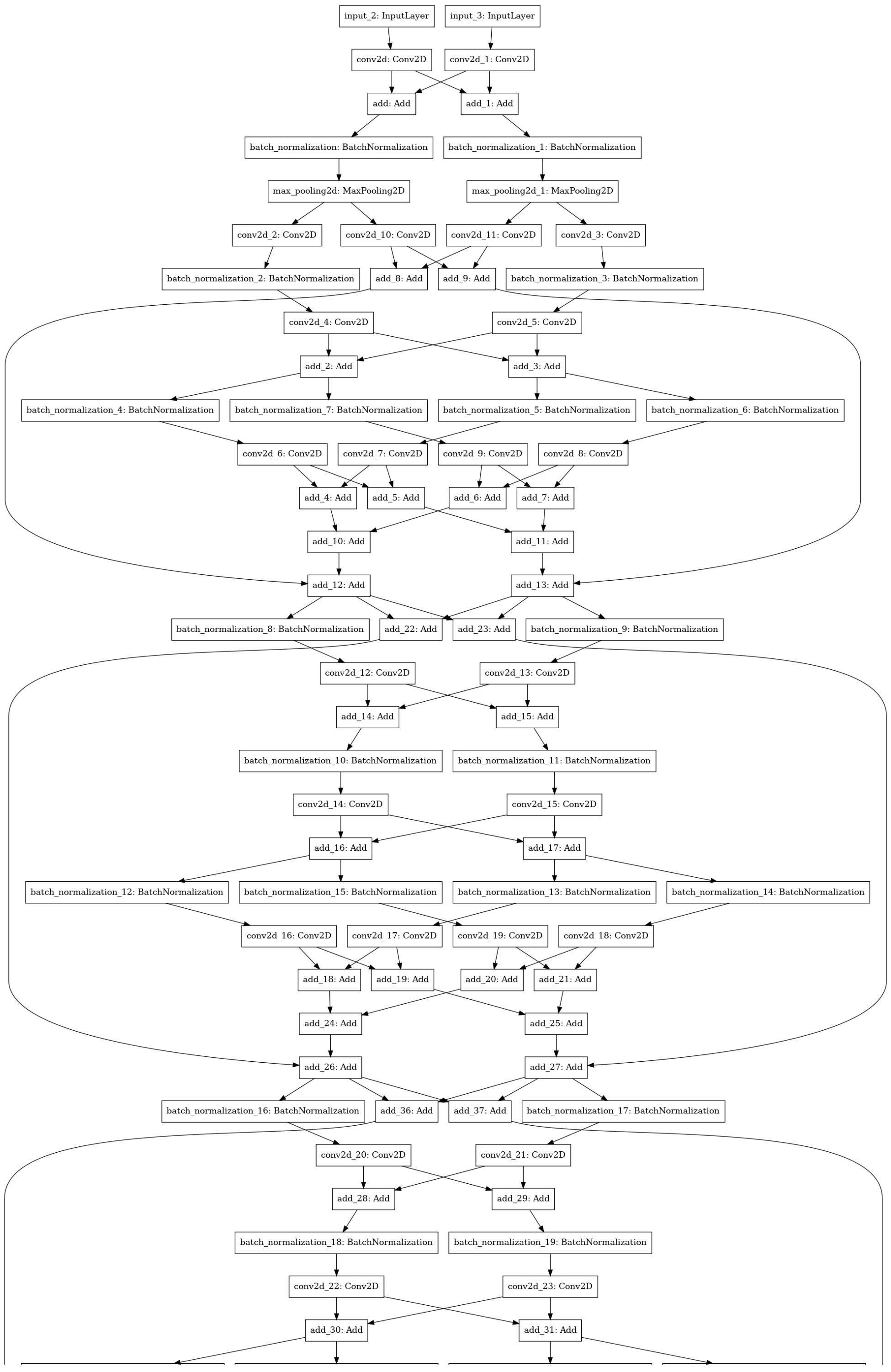
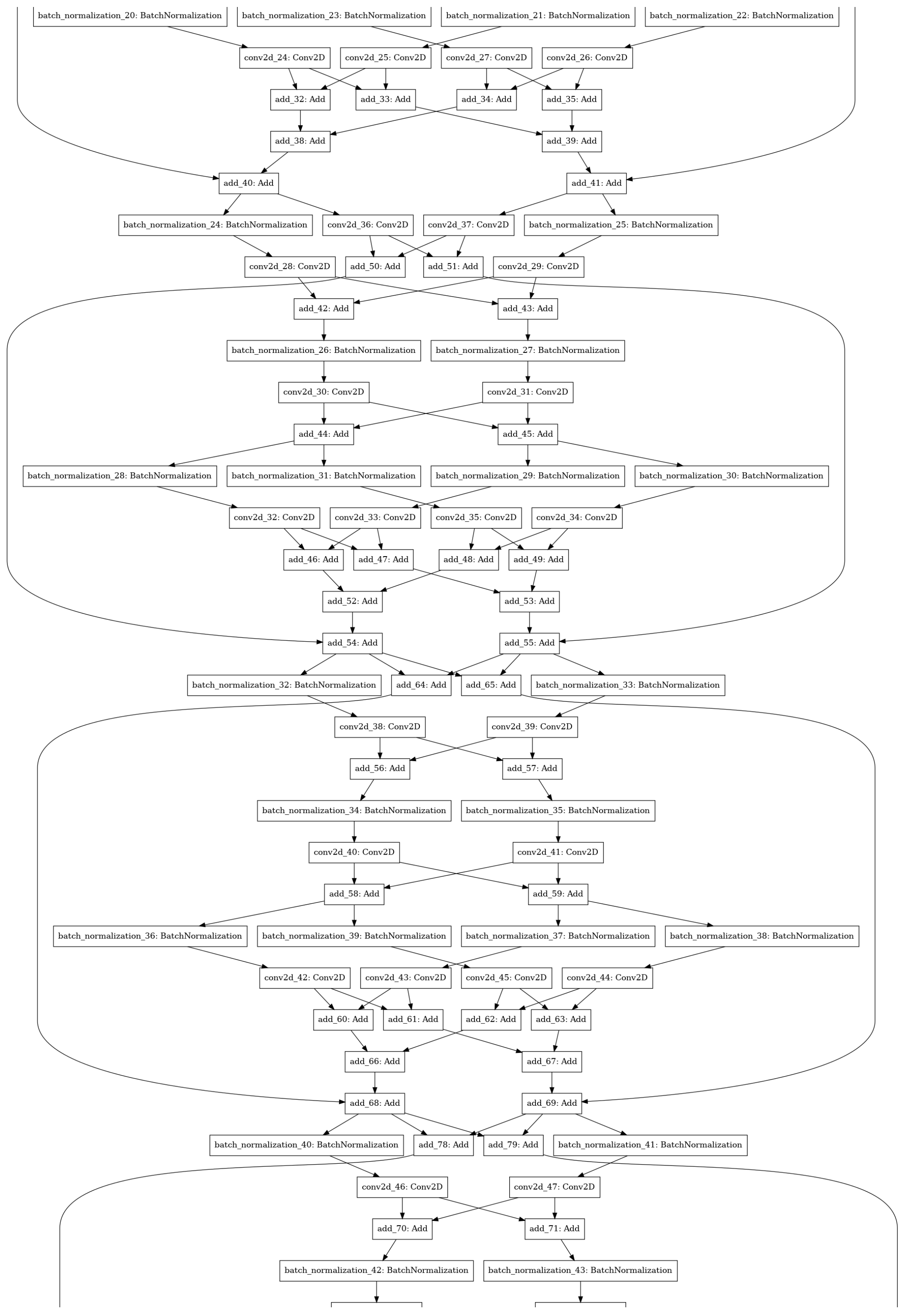
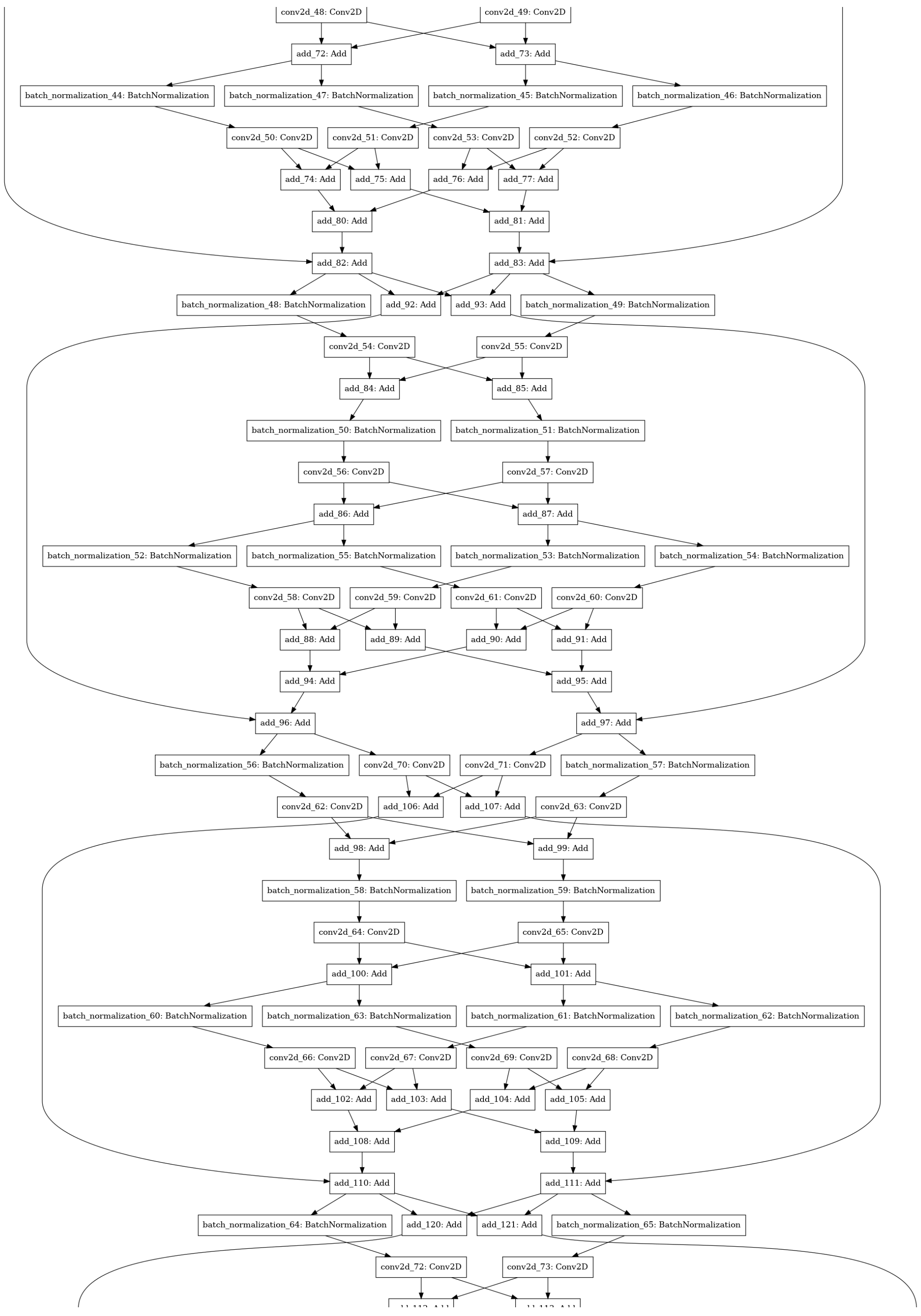

average_58: Average
batch_normalization_64: BatchNormalization
average_59: Average
batch_normalization_65: BatchNormalization
separable_conv2d_56: SeparableConv2D
separable_conv2d_57: SeparableConv2D
average_60: Average
batch_normalization_66: BatchNormalization
average_61: Average
batch_normalization_67: BatchNormalization
separable_conv2d_58: SeparableConv2D
separable_conv2d_59: SeparableConv2D
average_62: Average
batch_normalization_68: BatchNormalization
average_63: Average
add_20: Add
batch_normalization_69: BatchNormalization
conv2d_10: Conv2D
add_21: Add
separable_conv2d_60: SeparableConv2D
separable_conv2d_61: SeparableConv2D
conv2d_11: Conv2D
average_65: Average
average_64: Average
batch_normalization_70: BatchNormalization
batch_normalization_73: BatchNormalization
batch_normalization_72: BatchNormalization
separable_conv2d_63: SeparableConv2D
separable_conv2d_62: SeparableConv2D
batch_normalization_71: BatchNormalization
average_66: Average
average_67: Average
batch_normalization_74: BatchNormalization
batch_normalization_75: BatchNormalization
max_pooling2d_6: MaxPooling2D
max_pooling2d_7: MaxPooling2D
add_22: Add
add_23: Add
separable_conv2d_64: SeparableConv2D
separable_conv2d_65: SeparableConv2D
batch_normalization_76: BatchNormalization
batch_normalization_77: BatchNormalization
separable_conv2d_66: SeparableConv2D
separable_conv2d_67: SeparableConv2D
average_68: Average
average_69: Average
batch_normalization_78: BatchNormalization
batch_normalization_79: BatchNormalization
concatenate: Concatenate
global_average_pooling2d: GlobalAveragePooling2D
dense: Dense

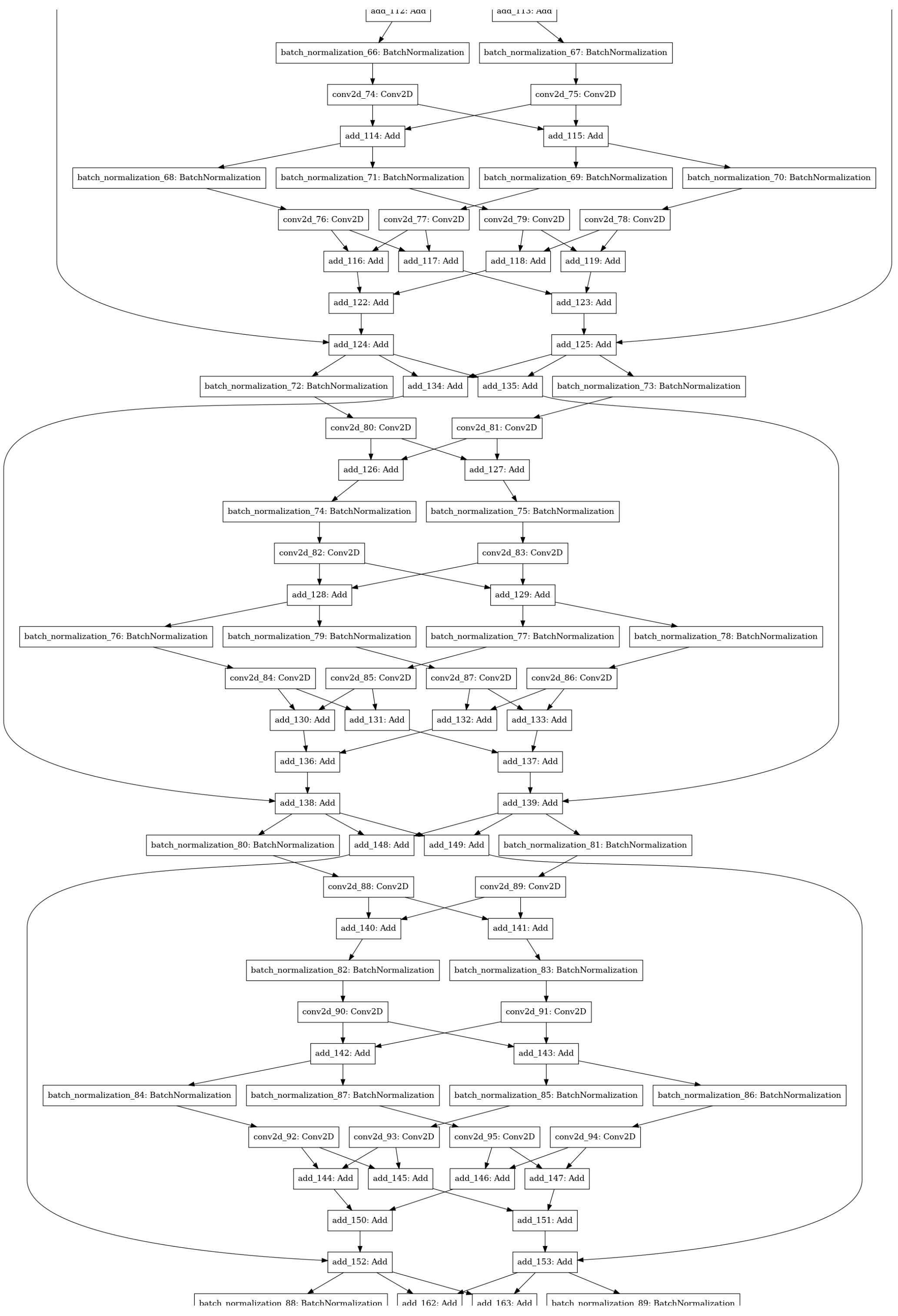
Figure 4: L-Xception.

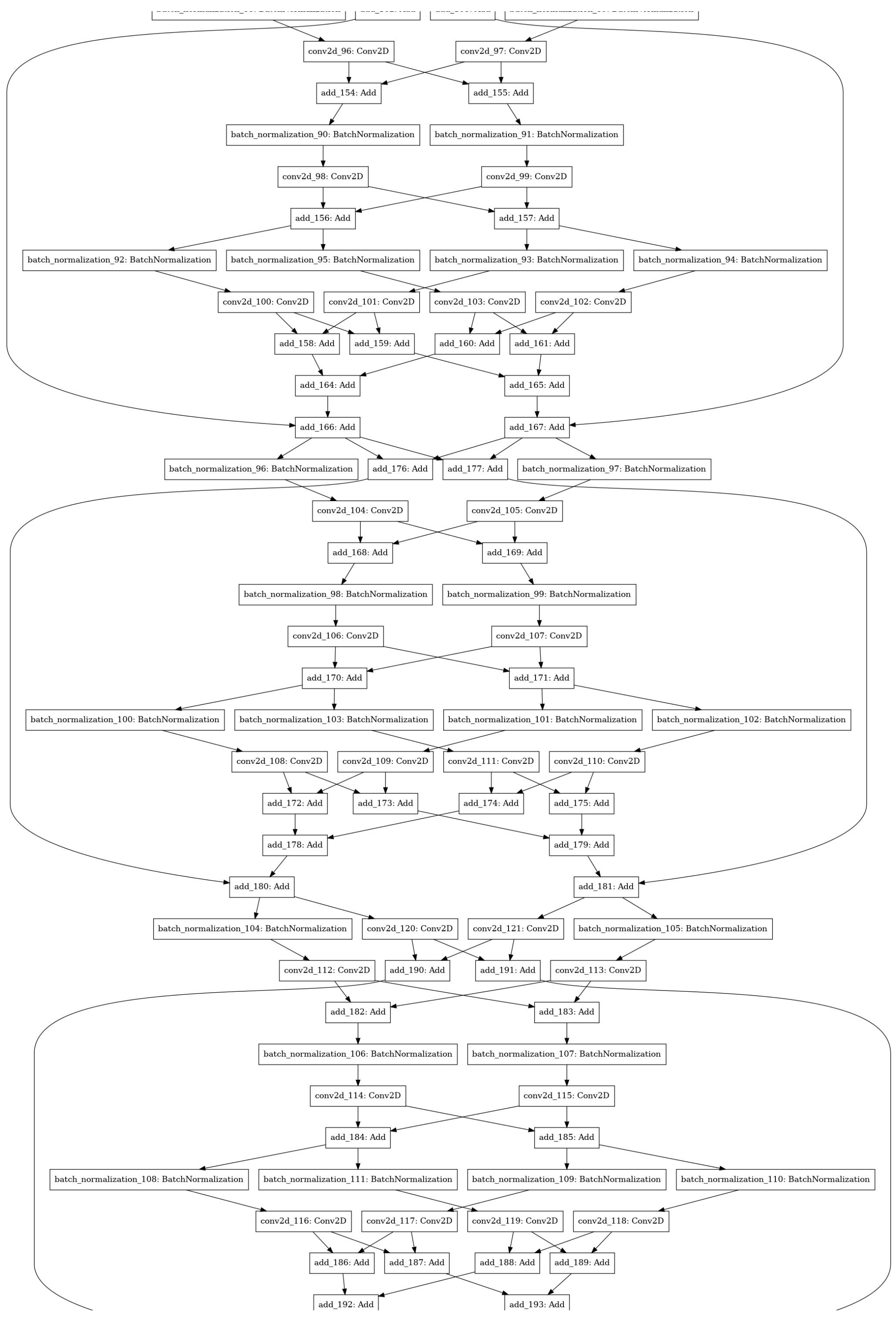

input_2: InputLayer
input_3: InputLayer
conv2d: Conv2D
conv2d_1: Conv2D
add: Add
add_1: Add
batch_normalization: BatchNormalization
batch_normalization_1: BatchNormalization
max_pooling2d: MaxPooling2D
max_pooling2d_1: MaxPooling2D
conv2d_2: Conv2D
conv2d_10: Conv2D
conv2d_11: Conv2D
conv2d_3: Conv2D
batch_normalization_2: BatchNormalization
add_8: Add
add_9: Add
batch_normalization_3: BatchNormalization
conv2d_4: Conv2D
conv2d_5: Conv2D
add_2: Add
add_3: Add
batch_normalization_4: BatchNormalization
batch_normalization_7: BatchNormalization
batch_normalization_5: BatchNormalization
batch_normalization_6: BatchNormalization
conv2d_6: Conv2D
conv2d_7: Conv2D
conv2d_9: Conv2D
conv2d_8: Conv2D
add_4: Add
add_5: Add
add_6: Add
add_7: Add
add_10: Add
add_11: Add
add_12: Add
add_13: Add
batch_normalization_8: BatchNormalization
add_22: Add
add_23: Add
batch_normalization_9: BatchNormalization
conv2d_12: Conv2D
conv2d_13: Conv2D
add_14: Add
add_15: Add
batch_normalization_10: BatchNormalization
batch_normalization_11: BatchNormalization
conv2d_14: Conv2D
conv2d_15: Conv2D
add_16: Add
add_17: Add
batch_normalization_12: BatchNormalization
batch_normalization_15: BatchNormalization
batch_normalization_13: BatchNormalization
batch_normalization_14: BatchNormalization
conv2d_16: Conv2D
conv2d_17: Conv2D
conv2d_19: Conv2D
conv2d_18: Conv2D
add_18: Add
add_19: Add
add_20: Add
add_21: Add
add_24: Add
add_25: Add
add_26: Add
add_27: Add
batch_normalization_16: BatchNormalization
add_36: Add
add_37: Add
batch_normalization_17: BatchNormalization
conv2d_20: Conv2D
conv2d_21: Conv2D
add_28: Add
add_29: Add
batch_normalization_18: BatchNormalization
batch_normalization_19: BatchNormalization
conv2d_22: Conv2D
conv2d_23: Conv2D
add_30: Add
add_31: Add

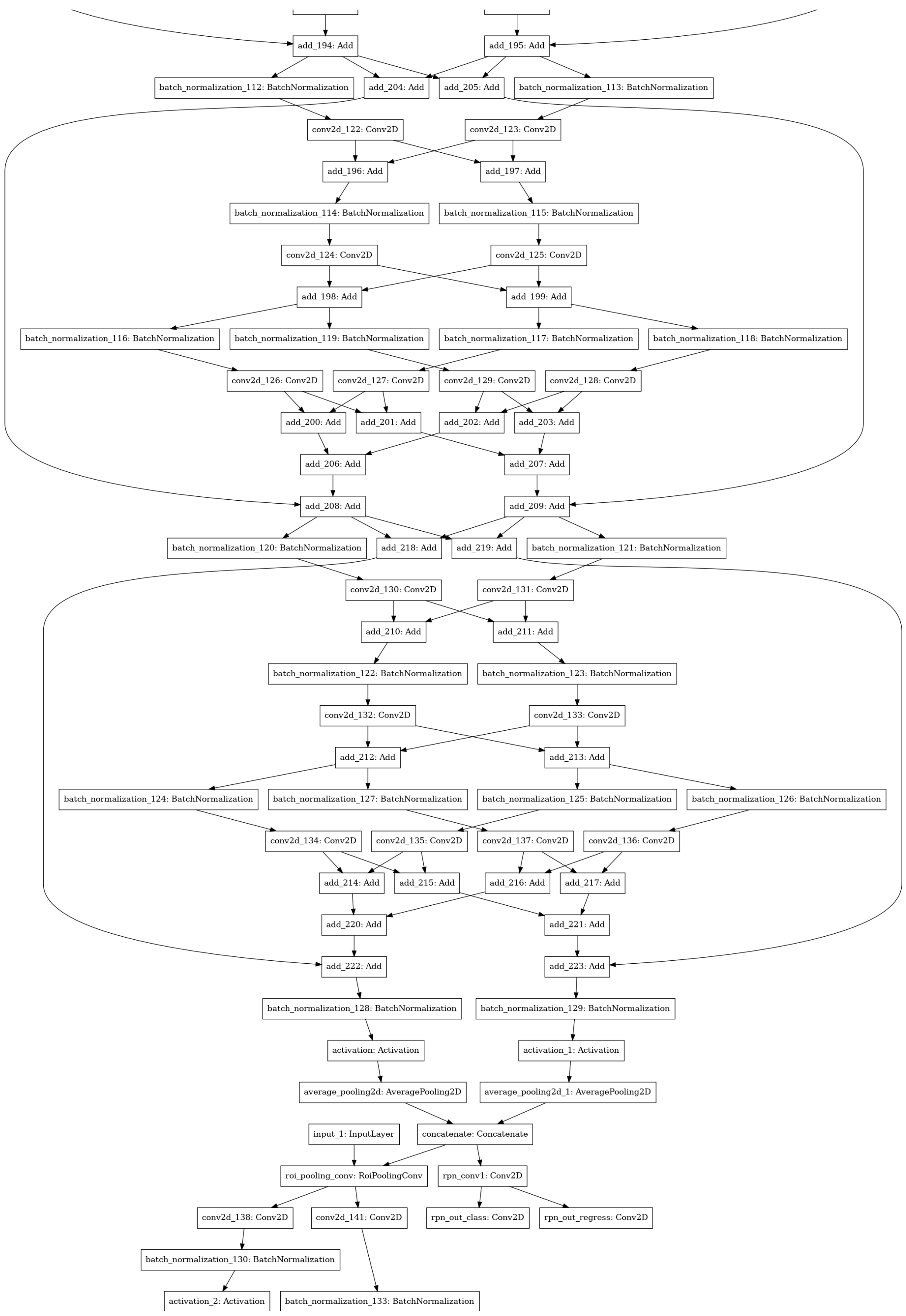


batch_normalization_20: BatchNormalization
batch_normalization_23: BatchNormalization
batch_normalization_21: BatchNormalization
batch_normalization_22: BatchNormalization
conv2d_24: Conv2D
conv2d_25: Conv2D
conv2d_27: Conv2D
conv2d_26: Conv2D
add_32: Add
add_33: Add
add_34: Add
add_35: Add
add_38: Add
add_39: Add
add_40: Add
add_41: Add
batch_normalization_24: BatchNormalization
conv2d_36: Conv2D
conv2d_37: Conv2D
batch_normalization_25: BatchNormalization
conv2d_28: Conv2D
add_50: Add
add_51: Add
conv2d_29: Conv2D
add_42: Add
add_43: Add
batch_normalization_26: BatchNormalization
batch_normalization_27: BatchNormalization
conv2d_30: Conv2D
conv2d_31: Conv2D
add_44: Add
add_45: Add
batch_normalization_28: BatchNormalization
batch_normalization_31: BatchNormalization
batch_normalization_29: BatchNormalization
batch_normalization_30: BatchNormalization
conv2d_32: Conv2D
conv2d_33: Conv2D
conv2d_35: Conv2D
conv2d_34: Conv2D
add_46: Add
add_47: Add
add_48: Add
add_49: Add
add_52: Add
add_53: Add
add_54: Add
add_55: Add
batch_normalization_32: BatchNormalization
add_64: Add
add_65: Add
batch_normalization_33: BatchNormalization
conv2d_38: Conv2D
conv2d_39: Conv2D
add_56: Add
add_57: Add
batch_normalization_34: BatchNormalization
batch_normalization_35: BatchNormalization
conv2d_40: Conv2D
conv2d_41: Conv2D
add_58: Add
add_59: Add
batch_normalization_36: BatchNormalization
batch_normalization_39: BatchNormalization
batch_normalization_37: BatchNormalization
batch_normalization_38: BatchNormalization
conv2d_42: Conv2D
conv2d_43: Conv2D
conv2d_45: Conv2D
conv2d_44: Conv2D
add_60: Add
add_61: Add
add_62: Add
add_63: Add
add_66: Add
add_67: Add
add_68: Add
add_69: Add
batch_normalization_40: BatchNormalization
add_78: Add
add_79: Add
batch_normalization_41: BatchNormalization
conv2d_46: Conv2D
conv2d_47: Conv2D
add_70: Add
add_71: Add
batch_normalization_42: BatchNormalization
batch_normalization_43: BatchNormalization


conv2d_48: Conv2D
conv2d_49: Conv2D
add_72: Add
add_73: Add
batch_normalization_44: BatchNormalization
batch_normalization_47: BatchNormalization
batch_normalization_45: BatchNormalization
batch_normalization_46: BatchNormalization
conv2d_50: Conv2D
conv2d_51: Conv2D
conv2d_53: Conv2D
conv2d_52: Conv2D
add_74: Add
add_75: Add
add_76: Add
add_77: Add
add_80: Add
add_81: Add
add_82: Add
add_83: Add
batch_normalization_48: BatchNormalization
add_92: Add
add_93: Add
batch_normalization_49: BatchNormalization
conv2d_54: Conv2D
conv2d_55: Conv2D
add_84: Add
add_85: Add
batch_normalization_50: BatchNormalization
batch_normalization_51: BatchNormalization
conv2d_56: Conv2D
conv2d_57: Conv2D
add_86: Add
add_87: Add
batch_normalization_52: BatchNormalization
batch_normalization_55: BatchNormalization
batch_normalization_53: BatchNormalization
batch_normalization_54: BatchNormalization
conv2d_58: Conv2D
conv2d_59: Conv2D
conv2d_61: Conv2D
conv2d_60: Conv2D
add_88: Add
add_89: Add
add_90: Add
add_91: Add
add_94: Add
add_95: Add
add_96: Add
add_97: Add
batch_normalization_56: BatchNormalization
conv2d_70: Conv2D
conv2d_71: Conv2D
batch_normalization_57: BatchNormalization
conv2d_62: Conv2D
add_106: Add
add_107: Add
conv2d_63: Conv2D
add_98: Add
add_99: Add
batch_normalization_58: BatchNormalization
batch_normalization_59: BatchNormalization
conv2d_64: Conv2D
conv2d_65: Conv2D
add_100: Add
add_101: Add
batch_normalization_60: BatchNormalization
batch_normalization_63: BatchNormalization
batch_normalization_61: BatchNormalization
batch_normalization_62: BatchNormalization
conv2d_66: Conv2D
conv2d_67: Conv2D
conv2d_69: Conv2D
conv2d_68: Conv2D
add_102: Add
add_103: Add
add_104: Add
add_105: Add
add_108: Add
add_109: Add
add_110: Add
add_111: Add
batch_normalization_64: BatchNormalization
add_120: Add
add_121: Add
batch_normalization_65: BatchNormalization
conv2d_72: Conv2D
conv2d_73: Conv2D


add_112: Add
add_113: Add
batch_normalization_66: BatchNormalization
batch_normalization_67: BatchNormalization
conv2d_74: Conv2D
conv2d_75: Conv2D
add_114: Add
add_115: Add
batch_normalization_68: BatchNormalization
batch_normalization_71: BatchNormalization
batch_normalization_69: BatchNormalization
batch_normalization_70: BatchNormalization
conv2d_76: Conv2D
conv2d_77: Conv2D
conv2d_79: Conv2D
conv2d_78: Conv2D
add_116: Add
add_117: Add
add_118: Add
add_119: Add
add_122: Add
add_123: Add
add_124: Add
add_125: Add
batch_normalization_72: BatchNormalization
add_134: Add
add_135: Add
batch_normalization_73: BatchNormalization
conv2d_80: Conv2D
conv2d_81: Conv2D
add_126: Add
add_127: Add
batch_normalization_74: BatchNormalization
batch_normalization_75: BatchNormalization
conv2d_82: Conv2D
conv2d_83: Conv2D
add_128: Add
add_129: Add
batch_normalization_76: BatchNormalization
batch_normalization_79: BatchNormalization
batch_normalization_77: BatchNormalization
batch_normalization_78: BatchNormalization
conv2d_84: Conv2D
conv2d_85: Conv2D
conv2d_87: Conv2D
conv2d_86: Conv2D
add_130: Add
add_131: Add
add_132: Add
add_133: Add
add_136: Add
add_137: Add
add_138: Add
add_139: Add
batch_normalization_80: BatchNormalization
add_148: Add
add_149: Add
batch_normalization_81: BatchNormalization
conv2d_88: Conv2D
conv2d_89: Conv2D
add_140: Add
add_141: Add
batch_normalization_82: BatchNormalization
batch_normalization_83: BatchNormalization
conv2d_90: Conv2D
conv2d_91: Conv2D
add_142: Add
add_143: Add
batch_normalization_84: BatchNormalization
batch_normalization_87: BatchNormalization
batch_normalization_85: BatchNormalization
batch_normalization_86: BatchNormalization
conv2d_92: Conv2D
conv2d_93: Conv2D
conv2d_95: Conv2D
conv2d_94: Conv2D
add_144: Add
add_145: Add
add_146: Add
add_147: Add
add_150: Add
add_151: Add
add_152: Add
add_153: Add
batch_normalization_88: BatchNormalization
add_162: Add
add_163: Add
batch_normalization_89: BatchNormalization


conv2d_96: Conv2D
conv2d_97: Conv2D
add_154: Add
add_155: Add
batch_normalization_90: BatchNormalization
batch_normalization_91: BatchNormalization
conv2d_98: Conv2D
conv2d_99: Conv2D
add_156: Add
add_157: Add
batch_normalization_92: BatchNormalization
batch_normalization_95: BatchNormalization
batch_normalization_93: BatchNormalization
batch_normalization_94: BatchNormalization
conv2d_100: Conv2D
conv2d_101: Conv2D
conv2d_103: Conv2D
conv2d_102: Conv2D
add_158: Add
add_159: Add
add_160: Add
add_161: Add
add_164: Add
add_165: Add
add_166: Add
add_167: Add
batch_normalization_96: BatchNormalization
add_176: Add
add_177: Add
batch_normalization_97: BatchNormalization
conv2d_104: Conv2D
conv2d_105: Conv2D
add_168: Add
add_169: Add
batch_normalization_98: BatchNormalization
batch_normalization_99: BatchNormalization
conv2d_106: Conv2D
conv2d_107: Conv2D
add_170: Add
add_171: Add
batch_normalization_100: BatchNormalization
batch_normalization_103: BatchNormalization
batch_normalization_101: BatchNormalization
batch_normalization_102: BatchNormalization
conv2d_108: Conv2D
conv2d_109: Conv2D
conv2d_111: Conv2D
conv2d_110: Conv2D
add_172: Add
add_173: Add
add_174: Add
add_175: Add
add_178: Add
add_179: Add
add_180: Add
add_181: Add
batch_normalization_104: BatchNormalization
conv2d_120: Conv2D
conv2d_121: Conv2D
batch_normalization_105: BatchNormalization
conv2d_112: Conv2D
add_190: Add
add_191: Add
conv2d_113: Conv2D
add_182: Add
add_183: Add
batch_normalization_106: BatchNormalization
batch_normalization_107: BatchNormalization
conv2d_114: Conv2D
conv2d_115: Conv2D
add_184: Add
add_185: Add
batch_normalization_108: BatchNormalization
batch_normalization_111: BatchNormalization
batch_normalization_109: BatchNormalization
batch_normalization_110: BatchNormalization
conv2d_116: Conv2D
conv2d_117: Conv2D
conv2d_119: Conv2D
conv2d_118: Conv2D
add_186: Add
add_187: Add
add_188: Add
add_189: Add
add_192: Add
add_193: Add



add_194: Add
add_195: Add
batch_normalization_112: BatchNormalization
add_204: Add
add_205: Add
batch_normalization_113: BatchNormalization
conv2d_122: Conv2D
conv2d_123: Conv2D
add_196: Add
add_197: Add
batch_normalization_114: BatchNormalization
batch_normalization_115: BatchNormalization
conv2d_124: Conv2D
conv2d_125: Conv2D
add_198: Add
add_199: Add
batch_normalization_116: BatchNormalization
batch_normalization_119: BatchNormalization
batch_normalization_117: BatchNormalization
batch_normalization_118: BatchNormalization
conv2d_126: Conv2D
conv2d_127: Conv2D
conv2d_129: Conv2D
conv2d_128: Conv2D
add_200: Add
add_201: Add
add_202: Add
add_203: Add
add_206: Add
add_207: Add
add_208: Add
add_209: Add
batch_normalization_120: BatchNormalization
add_218: Add
add_219: Add
batch_normalization_121: BatchNormalization
conv2d_130: Conv2D
conv2d_131: Conv2D
add_210: Add
add_211: Add
batch_normalization_122: BatchNormalization
batch_normalization_123: BatchNormalization
conv2d_132: Conv2D
conv2d_133: Conv2D
add_212: Add
add_213: Add
batch_normalization_124: BatchNormalization
batch_normalization_127: BatchNormalization
batch_normalization_125: BatchNormalization
batch_normalization_126: BatchNormalization
conv2d_134: Conv2D
conv2d_135: Conv2D
conv2d_137: Conv2D
conv2d_136: Conv2D
add_214: Add
add_215: Add
add_216: Add
add_217: Add
add_220: Add
add_221: Add
add_222: Add
add_223: Add
batch_normalization_128: BatchNormalization
batch_normalization_129: BatchNormalization
activation: Activation
activation_1: Activation
average_pooling2d: AveragePooling2D
average_pooling2d_1: AveragePooling2D
input_1: InputLayer
concatenate: Concatenate
roi_pooling_conv: RoiPoolingConv
rpn_conv1: Conv2D
conv2d_138: Conv2D
conv2d_141: Conv2D
rpn_out_class: Conv2D
rpn_out_regress: Conv2D
batch_normalization_130: BatchNormalization
activation_2: Activation
batch_normalization_133: BatchNormalization

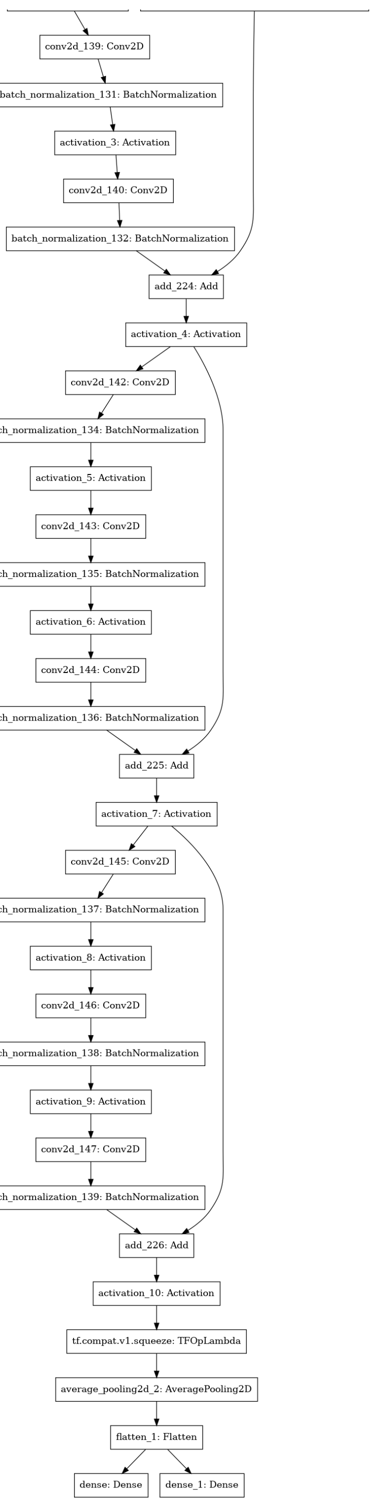

conv2d_139: Conv2D
batch_normalization_131: BatchNormalization
activation_3: Activation
conv2d_140: Conv2D
batch_normalization_132: BatchNormalization
add_224: Add
activation_4: Activation
conv2d_142: Conv2D
batch_normalization_134: BatchNormalization
activation_5: Activation
conv2d_143: Conv2D
batch_normalization_135: BatchNormalization
activation_6: Activation
conv2d_144: Conv2D
batch_normalization_136: BatchNormalization
add_225: Add
activation_7: Activation
conv2d_145: Conv2D
batch_normalization_137: BatchNormalization
activation_8: Activation
conv2d_146: Conv2D
batch_normalization_138: BatchNormalization
activation_9: Activation
conv2d_147: Conv2D
batch_normalization_139: BatchNormalization
add_226: Add
activation_10: Activation
tf.compat.v1.squeeze: TFOpLambda
average_pooling2d_2: AveragePooling2D
flatten_1: Flatten
dense: Dense
dense_1: Dense

Figure 5: L-FRCNN.