UNIVERSITY OF SOUTHAMPTON

Deep Cascade Learning

by

Enrique S. Marquez

A thesis submitted in partial fulfilment for the degree of Doctor of Philosophy

in the

Faculty of Engineering and Physical Sciences
Electronics and Computer Science

June 2019

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Enrique S. Marquez

Deep Learning has demonstrated outstanding performance on several machine learning tasks. These results are attributed to training very deep networks on large scale datasets. In this thesis we investigate training models in a layer-wise fashion. We quantify performance and discuss the advantages of using such training algorithms on computer vision and signal processing tasks.

Inspired by the Cascade Correlation algorithm, which is a growing neural network that iteratively learns artificial neurons, we developed a supervised layer-wise training algorithm, which we name Deep Cascade Learning. Our methodology takes as input the architecture to train and splits the model in submodels, where each iteration trains only one layer of the network. The feature representation gets more robust as layers are stacked. Moreover, the algorithm provides training complexity reduction while preserving competitive results in comparison with state-of-the-art end to end training. We demonstrate these advantages on multiple benchmark datasets.

Given that Deep Cascade Learning trains models from scratch successfully, we also look at layer-wise methods to transfer features from a large base dataset, to a smaller target dataset. This is particularly useful when the target dataset cannot be used to train a model from scratch due to lack of data. This second algorithm, which we named Cascade Transfer Learning (CTC), yields similar memory advantages to Deep Cascade Learning, and enables minimal computational complexity for feature transfer. In addition, CTC provides competitive results in comparison with other transfer learning approaches.

Finally, we further explore the scalability of Deep Cascade Learning by executing it on a multi-variate time series classification task. Such tasks include predicting human activities from body-worn sensors. Deep Cascade Learning can be used to reduce the training time of these models, opening up the possibility of online training on smart devices.

Contents

Declaration of Authorship

I, Enrique S. Marquez , declare that the thesis entitled and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself
- parts of this work has been published, refer to Section 1 for more details about this publication.

Signed:.....

Date:.....

Acknowledgements

First and foremost I would like to thank my parents for their emotional and financial support during this academic journey.

Secondly, I would like to thank my supervisors Dr. Jonathon Hare and Prof. Mahesan Niranjan for encouraging me to take the PhD. Thanks to them I have acquired invaluable knowledge that will help developing my career.

I extend my gratitude to all the people I have met these past years. Their time to discuss relevant ideas regarding my PhD thesis was priceless.

Finally, I would like thank the VLC Research Group for offering an outstanding work environment. Their daily discussions, recommendations, advice, and activities made my PhD a very enjoyable journey.

Chapter 1

Context & Contributions

Recent developments in machine learning have shown state-of-the-art results by applying deep learning to complex datasets. The result is deep differentiable networks capable of merging the standard machine learning pipeline (feature extraction and classification) into a single training algorithm. Deep networks learn optimal weights by minimizing a loss function. These weights are constrained in order to generate robust and invariant features. The classifier is directly connected to the top hidden layer of these set of weights (layers of feature extraction), which allows the network to optimally tune the synergy between features and classifier.

The traditional deep learning training algorithm uses backpropagation to update the weights of the hidden layers [108]. In the models, layers are updated as the gradient is propagated from bottom (last layer) to top (first layer). The magnitude of the gradients on early layers is affected by the multiplicative effect of the chain rule. In other words, the magnitude will always decrease for layers closer to the input. This well-known problem, often described as vanishing gradient, is tackled extensively by the literature [70, 90, 45]. However, approaches such as ReLU [90] and residual connections [45] alleviate the vanishing gradient problem but do not totally overcome it [125]. The vanishing gradient problem is further discussed in Chapter 2 and 4.

The aim of this thesis is to explore supervised layer-wise training of deep learning models. We present a novel Deep Cascade Learning algorithm and demonstrate its effectiveness on multiple tasks. We motivate our algorithm on the premise that early layers do not learn meaningful representations of the data, and that the robustness of the features is built in later stages of the network. By applying a layer-wise training, a classifier is directly connected to every layer, which ultimately increases the gradient magnitude.

Through this work we propose a novel training algorithm to sequentially learn representations of the data. We quantify performance and robustness of our method on image classification, time series classification and a novel approach to apply transfer learning on the image classification domain.

Contributions

The main contributions of this work are as follows:

Deep Cascade Learning

- **Complexity advantages**. The algorithm itself is capable of achieving similar performance to traditional training while requiring a fraction of the time. As opposed to end to end training, the computational cost decreases as the model gets deeper.
- **Performance**. Our algorithm obtains competitive results with the state-of-the-art when fine-tuning the network after cascading. However, there is a computational cost associated with this fine-tuning.
- **Applications**. We show quantitative analysis on image classification using the CIFAR benchmark dataset, which contains 10 or 100 classes in its two variants. In addition, we quantify the performance of cascading 1D Convolutional Networks on a multi-variate time series classification problem.

Cascade Transfer Learning

- **Memory Complexity**. Traditional transfer learning includes a fine-tuning stage after training the classifier on the last layer. Our approach trains layer by layer which only requires to store one block at the time drastically decreasing the complexity of the training. This enables the users to apply transfer learning using less hardware than the required to train the network on the base dataset.
- **Performance**. The algorithm generalizes better on multiple image classification tasks for relatively small networks, and achieves competitive results on very deep networks.
- **Scalability**. We tested our algorithm on five datasets and showed its applicability to multiple target datasets.
- **Resource Efficient Prediction**. The algorithm yields several classifiers with different accuracy. Early classifiers can be used to make predictions on 'easy' inputs, allowing the network to stop inference at mid stages.

Thesis structure

The thesis is structured as follows:

Chapter 2 covers the literature on Deep Learning and neural networks. We discuss the methodology of computer vision and signal processing in detail along with the most successful architectures to date. The Chapter starts with early approaches on neural networks including perceptrons and MLPs, and goes through state-of-the-art deep learning models with dense and residual connections [45]. We also revise temporal neural networks that are applied to time series data.

Chapter 3 explores the literature on layer wise training. The section includes the Cascade Correlation algorithm [28] which presents an early approach to progressively train multi-layer perceptrons. Additionally, we explore other recent approaches to iteratively learn layers using unsupervised algorithms [48, 68]. Finally, we discuss a very powerful architecture to progressively increase the size of a convolutional generative model for image generation [61].

Chapter 4 presents the approach and shows quantitative analysis of our Deep Cascade Learning algorithm on two image classification tasks. The algorithm shares advantageous properties with other layer-wise approaches while being executed on state-of-the-art network architectures.

Chapter 5 analyses a novel transfer learning approach where pretrained layers are sequentially adapted using a cascade-like algorithm. We show competitive results on three target datasets and use ImageNet dataset from the Large Scale Visual Recognition Competition [109] (ILSVRC) as the base dataset. Cascade Transfer Learning (CTL) yields complexity advantages and allows applying transfer learning on any pretrained network without increasing the memory requirements. The algorithm is most successful on very deep networks. For CTL, depth is not correlated with the required training memory, as opposed to traditional fine-tuned transfer learning. Additionally, CTL can find the stage at which the network transfers the features better to the target dataset.

Chapter 6 discusses networks for Human Activity Recognition (HAR). This problem includes sensor data, and can be seen as a multivariate time series classification problem. We present results on residual networks for HAR on three datasets. In addition, we show that Deep Cascade Learning is scalable to 1D Residual Networks and is not limited to 2D CNNs.

Chapter 7 summarizes the studies and findings developed during this thesis. Finally, we discuss future work covering layer wise training, as well as potential scalability of our algorithms to other applications.

Publications

- Published Paper at IEEE Transactions on Neural Networks and Learning Systems. Deep Cascade Learning.

  Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2018

Chapter 2

Deep Learning

Deep Learning can be seen as a set of machine learning methods that allow training deep neural networks successfully. A deep neural networks (DNN) is such model that cannot be naively trained with only backpropagation. Further regularisation and optimisation methods have to be included in order to improve generalisation of DNNs. In addition, GPU acceleration and automatic differentiation has allowed deep networks to learn from large scale complex data. Before the deep learning breakthrough, neural networks were outperformed by traditional approaches (e.g SVMs, Random Forest) due to their poor scalability to larger problems and lack of understanding.

Rosenblatt [106] successfully developed the first artificial neuron called the Perceptron [106]. This artificial neuron holds some properties that are still applicable to modern neural networks. However, some ideas did not catch up and were discarded by subsequent studies. Such ideas include training multi-layer perceptrons (MLP) with a fixed randomized early layers, because of lack of knowledge on how to update these early weights. The single perceptron was limited to its linear boundary, thus, making it impossible to learn non-linear functions, such as the XOR function [87]. The Backpropagation algorithm was presented as a way of learning multiple perceptrons by updating the weights using its derivatives and the chain rule [107]. This learning procedure greatly boosted the capabilities of neural networks, which enabled them to learn more complex functions. Besides training a stack of perceptrons end to end, Fahlman and Lebiere [28] presented an adaptive architecture to iteratively train multiple perceptrons. This 'Cascade Correlation' algorithm includes several techniques with similarities across modern deep learning. Nevertheless, neural networks were computationally expensive with complex hidden mechanics; for this reason, they were put aside by the community and were taken over by statistical approaches to machine learning.

Improvements in neural network architectures were still been made after MLPs were developed. In the computer vision field, Fukushima [33] presents the Neocognitron, an artificial neural network inspired by discoveries on the visual primary cortex. The

Neocognitron presented a few concepts that were then applied into more robust visual networks, such concepts include feature integration and receptive field. Subsequently, LeCun et al. [71] proposed the first Convolutional Neural Network (CNN) named LeNet-5. The architecture contributed key concepts that are applicable to modern state-of-the-art CNNs, such as weight sharing, and downsampling. After LeNet-5, the community followed a standard pipeline of classifying handcrafted features to tackle vision problems due to the complexity of CNNs, which made almost impossible to train on large scale datasets.

Similarly, the community explored the possibility of using recurrent units to capture temporal patterns in the data [86]. Recurrent Neural Networks (RNNs) allows artificial neurons to process sequences of inputs, hence, extracting temporal information from the input sequences. Hochreiter and Schmidhuber [49] presented the Long short term memory (LSTM), which is an architecture to hold or forget information across the time axis. LSTMs showed better performance on several tasks by extracting short and long patterns. This improvement enabled recurrent architectures to overcome vanilla RNNs. Their applicability goes from handwriting recognition [39] to speech applications [74].

The decay in neural network research was a result of the training complexity of these networks. Recent developments in processing power and GPU acceleration has drastically decreased the training time. Nowadays, these networks are not only computationally feasible, but we are now capable of increasing the number of parameters to learn more complex representations. In addition, with the digital revolution the availability of the data has dramatically increased. The machine learning community can now gather thousands of data points and iteratively train these networks using batch training.

Deep networks, however, still hold several drawbacks in contrast with traditional machine learning approaches (e.g SVM and Random Forest). One of the main criticisms is the interpretability of the models. It is not straightforward to quantify the internal mechanics of deep networks, which leaves the open question, how do deep networks actually learn representative features? For this reason the community has been researching around understanding the "black box" that is deep learning. Shwartz-Ziv and Tishby [113] conclude that DNN find efficient representations by approximating minimal sufficient statistics in the sense of the information bottleneck. In addition, Lin et al. [75] aims to unravel deep networks through physics by exploring frequently presented properties that are similar in both domains, such as symmetry, locality, compositionality, and polynomial log-probability.

Besides the interpretability problem, deep networks suffer from degradation of the back-propagated gradients. Vanishing gradients in early layers generates smaller updates of weights, which do not allow the network to learn meaningful representations on early stages [45]. To define the vanishing gradient problem, we make use of a small network with four hidden layers. The parameters of this model are defined as: weights

$\mathbf{W} = w_1, w_2, w_3, w_4$ (for the purpose of this definition the shape/nature of the weights is irrelevant), biases $\mathbf{B} = b_1, b_2, b_3, b_4$, hidden layers outputs $\mathbf{Z} = z_1, z_2, z_3, z_4$, the inputs to the hidden layers are defined as x_1, x_2, x_3, x_4 and final output y. Hence, we can define the output of a layer as $z_i = f(w_i, x_i) + b_i$, where f is the function between inputs and weights (e.g linear for dense layers, or convolutional for convolutional layers). z_i is then evaluated using the predefined activation function σ. If we apply the backpropagation using the chain rule to obtain the first hidden layer gradients we get, $\frac{\partial y}{\partial w_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial y}{\partial z_4}$. If we analyse the previous equation we realise that to obtain the gradients of the first layer, we have to multiply the derivative of the activation function σ several times. Hence, if $\sigma' < 1$, the resulting value of multiplying several number between $0 - 1$ will be very small. In addition, the more layers we use, the more multiplications are performed in the chain rule, and the smaller the resulting value for early layers will be. This can be alleviated by using activation functions with larger derivatives such as Rectified Linear Units [90] (further discussed in later sections of this chapter). However, we can spot that to obtain $\frac{\partial y}{\partial w_1}$ it also has to be included multiplications of the weights w_i, which are very often initialised between $0-1$. This also has an effect on the vanishing gradient problem, thus changing the activation function does not fully overcome this issue.

The training of deep networks has been optimized over the past few years, in order to enhance their stability, performance, and complexity. The methodology includes:

- **Regularization methods**. In order to avoid overfitting the training data. Such methods include: dropout [117], constraints to the loss function (e.g weight decay and L_1 norm [36]) and batch normalization [55].
- **Complexity reduction**. Methods to reduce training complexity such as batch training [48, 67]; and methods to reduce the cost of inference after training, such as pruning or binarizing the weights [100, 42].
- **Optimization**. The latest algorithms include optimization steps to speed up the convergence of the network (e.g cross-entropy loss function for classification and Adam optimizer [63]). These methods are further discussed on the next section.
- **Initialization**. Better initial weights leads to faster and more efficient convergence [46]. We discuss this in more details in the next section.
- **Architectures**. The community has explored multiple architectures to learn better representations. Most recently, residual connections have been a trend towards better learning, we discuss this into depth in Section 2.2. This improvement is developed as a way of circumventing the vanishing gradient problem.

2.1 Relevant concepts & state-of-the-art in DL

In this section we discuss into detail several concepts used during this thesis. We describe a set of commonly applied architectures as well as state-of-the-art methodology to train these networks.

- **Convolutional Neural Networks** (CNNs). These models are motivated by studies around animals visual cortex [53]. Neocognitron [33] was a first approach to emulate the visual system. However, it failed to incorporate with key aspects of the cortex, such as its hierarchical structure and translational invariance. LeCun et al. [71] developed LeNet-5 as first successful CNN trained to classify handwriting letters and digits. CNNs are feedforward networks in which the parameters are filters or kernels instead of linear weights. Each filter produces one feature by convolving it with the input. Theoretically the feature is translation invariant, thus allowing the network to search for the same pattern across the spatial input. The stack of convolutional layers is known as a feature extractor, often referred as an encoder. The MLP on the last stage of the network is known as the classifier. The model contains activation functions to learn non-linear features. AlexNet [67] is a CNN architecture trained on GPUs, which made feasible the training of these networks on large scale datasets. This architecture is further discussed on the Section 2.2. The entire network is differentiable and can be trained using backpropagation and Stochastic Gradient Descent (SGD) to minimize the loss function.

- **Recurrent Neural networks** (RNNs). In some problems the patterns to capture are not in the spatial domain, but rather in the temporal domain. Other applications of these networks may include sequence analysis and natural language processing. RNNs aim to extract such patterns using recurrent units to "hold" information from previous inputs. This recurrent procedure contains an internal state to process sequence of inputs, hence, preserving or forgetting features from past data. The vanilla RNN [129] is known to be capable of finding short term patterns. For the case when long term dependencies are necessary, Hochreiter and Schmidhuber [49] presents the Long Short Term Memory Networks (LSTMs). Derivatives of LSTMs, such as Gated Recurrent Unit(GRU) [15], have been explored extensively in the literature. These networks have produced state-of-the-art results on speech [1, 64] and text [119] problems. RNNs are less parallelisable than CNNs, computationally more complex, and not easily compressed [5].

- **Regularization**. In Deep Learning there are several techniques to avoid overfitting. Srivastava et al. [117] developed Dropout to prevent deep networks from overfitting the training data. It is a simple yet powerful method which is applied to the training routine. On every iteration of training, dropout "deactivates" some

units in the specified stage with a fixed probability (typically $p = 0.5$). By applying this, the network not only prevents complex co-adaptations on the training set, but also decreases the variance (high variance induces is reflected as overfitting the training set) by emulating an average of smaller sub-networks. Many years earlier, Fahlman and Lebiere [28] mentioned an insight to dropout when applying the cascade correlation algorithm, and stated that their algorithm generalized better when ignoring some weights during training. Constraints to the loss function, such as l_1 (which induces sparsity and feature selection) and l_2 norm are also applicable as regularization methods. Moreover, Batch Normalization [55] decreases the covariance shift of the features by learning a normalization function at intermediate stages of the network.

- **Rectified Linear Activation Function**. ReLU [90], in the machine learning context, is an activation function motivated to address the vanishing gradient problem. This is as a result of the derivative of the function ReLU, which is 1. In addition, computing its function and derivative is computationally more efficient than traditional activations, such as Sigmoid and hyperbolic tangent. However, ReLU still holds a "dying units" problem, where neurons are often driven to inactive states and cannot be brought back due to its zero gradient to the left of the zero [18]. Several approaches have been used to avoid having dead neurons on early stages of the training algorithm. Leaky ReLU [80] uses small slope on the negative axis of the function, this allows inactive neurons to be back to activate state since its derivative has a small value instead of zero. Other activation functions based on ReLU have been proposed as ways of better feature learning. For example, Parametric ReLU [46] uses a similar method as Leaky ReLU, the main difference remains in the slope, which is adaptively learnt and included in the backpropagation algorithm. Equations 2.1, 2.2, 2.3 show the mathematical operation of ReLU, Leaky ReLU (a is fixed) and PReLU (λ is learnt).

$$ReLU(x) = \max(0, x) \tag{2.1}$$

$$LeakyReLU(x) = \max(x, ax) \tag{2.2}$$

$$PReLU(x) = \max(x, \lambda x) \tag{2.3}$$

- **Optimization and loss functions**. Convergence of deep networks can be a complex problem and might require several forward passes of the data. In order to speed up training and ensure convergence, the community has developed optimization algorithms. Popular approaches include adaptive learning rates which consider the first and second moments of the gradients [27, 63, 134]. Depending on the problem, the loss function can be adapted to further enhance the training.

For example, on multi-class datasets one might apply a naïve loss function, such as Mean Square Error, however, using a Softmax function with CrossEntropy loss may squash the output into a probability vector and more rapidly find a minimum with desirable properties.

- **Initialization**. Arbitrarily initializing the weights can damage the network's convergence. Recent developments have found correlation between the number of parameters in the model and its 'ideal' initialization. He et al. [46] compares multiple initialization methods and presents a novel method to randomly initialise the weights from a fix distribution. Kaiming's initialization [46] considers the activation function of the network, for ReLU networks they concluded that the weights should be initialised from a Gaussian distribution with zero mean and $\sqrt{2/n_l}$ standard deviation, where n_l represents the number of parameters of layer l. These methodology has been recently adapted to modern deep learning networks, although, its motivation comes from early studies in initialising neural networks

2.2 Deep Learning for Computer Vision

CNNs have achieved state-of-the-art results on multiple visual learning tasks. Such tasks include image/object classification [67, 45], object detection [121], biometrics [122], segmentation [78], and medical image processing [62]. This breakthrough comes from training networks from scratch, as well as transferring the feature extractor from one large scale dataset to a less complex domain [67, 132]. This is known as transfer learning and is explained into details in later sections.

The convolution operator is defined as $y_{ij} = \sum_{a=0}^{k} \sum_{b=0}^{k} w_{ab}^{t} x_{(i+a)(j+b)}$ where y_{ij} is the output value at index i and j, w is the (for most computer vision cases) square filter with size k, and x is the input image. For multichannel cases (e.g RGB images, three channels with $C = 3$), the filters are multidimensional with shape (C, k, k) and can be unravelled as a linear combination of multiple 2D filters to generate a single feature map per multidimensional filter. In order to generate more than one feature, the layer may contain several multidimensional filters, hence the shape of the convolutional weights of a layer is defined as (N, C, k, k) where N is the number of "3D" filters to be learnt. The operation does not work on the whole input like in fully connected layers, instead it computes the operation on one fixed sized region of the image (receptive field) at the time using shared parameters. This generates robust translational invariant features, but does not allow these features to capture patterns at multiple scales. Thus, to make it more robust to scale, pooling operators are often used to downsample the features and learn filters at multiple scales. Commonly used pooling operators include Max Pooling, which takes the values of highly activated neurons and discards those with low activation value; and Average Pooling, which takes the average of the activations. Both use a

sliding window approach on fixed receptive field with a constant step (stride). Other pooling operators have been proposed to tackle different issues in the learning of features [37, 44]. Recent architectures do not use pooling operators, instead the downsampling can be learnt by a strided convolution [116]. It is intuitive to think that learning the downsampling function would be more beneficial than having a fixed operator, however, it adds some extra parameters to the network. The input to these layers are usually padded with zeros to preserve the dimensionality. Batch normalization is also included into the set of learnable layers of the CNNs. It aims to decrease the variance of the hidden features (covariance shift) by learning normalization parameters (mean and standard deviation) at mid stages of the network. Nevertheless, CNNs architectures might not use all the techniques listed, but a subset of them. Features are flattened on the last stage of the feature extractor in order to connect the classification device, which is often an MLP to make the network differentiable end to end.

LeNet-5 [71] introduces key concepts to train CNNs. It contains two convolutional layers with kernels of size $(6, 1, 5, 5)$ and $(16, 6, 5, 5)$ and downsample operations (2 by 2 average pooling) after every convolutional layer. The classification on that model is performed by two fully connected layers with 120 and 84 neurons each. After LeNet-5, the community followed more traditional approaches to tackle vision problems. This decay in CNN research was a result of the complexity of the training of these networks.

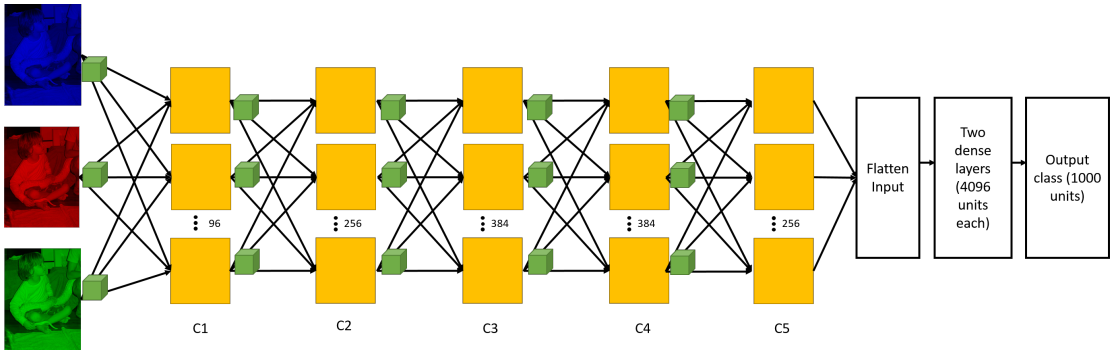

FIGURE 2.1: AlexNet CNN applied to ImageNet, input image RGB 224×224 and outputs the class. C1-5 are convolutional layers. C1, C2, and C5 have MaxPooling layers after the convolutions. The dimensionaly of the filters are: C1 $96 \times 11 \times 11$ (channels, rows, and columns of filter), C2 $256 \times 5 \times 5$, C3 $384 \times 3 \times 3$, C4 $384 \times 3 \times 3$, and C5 $256 \times 3 \times 3$.

The breakthrough of CNNs came with AlexNet [67] in 2012. Krizhevsky et al. [67] introduced a setup to train CNNs with GPU acceleration with a set of methods to reduce overfitting. They applied a CNN to the ImageNet challenge, which is a large scale visual classification competition containing 1.5M images and is further explained in Section 2.2.3. AlexNet won the 2012 ImageNet competition by a margin of 8% top-5 classification accuracy (see Section 2.2.2 for details on the metric). The algorithm in second place extracted SIFT descriptors using Fisher vectors and classified them with Support Vector Machines (SVMs) [110]. The network included dropout on the layers of

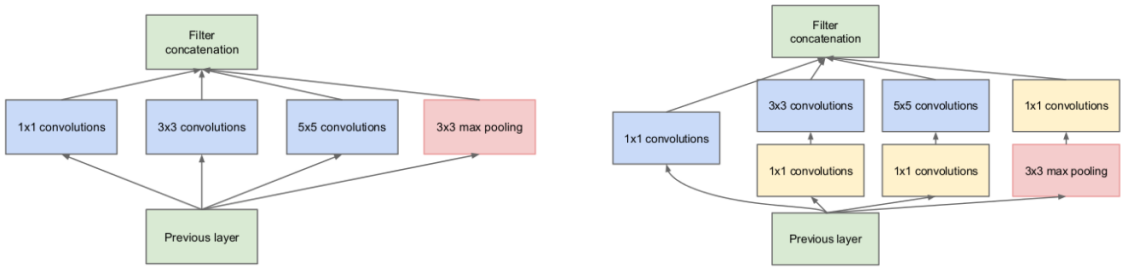

FIGURE 2.2: Inception modules used in GoogLeNet. (left) Naive approach, computationally untraceable, (right) Bottleneck approach with layers to reduce the dimensionality of the feature maps. Figure taken from the original paper [120].

the classifier, which helped avoid the network overfitting the training data. Dropout also increases the number of epochs necessary to train the models. In addition, ReLU was applied in every hidden layer as well as Softmax on the output layer. MaxPooling was introduced under this context to downsample the data after every convolution. Data Augmentation was included in Alex's Net methodology to generate more images from the raw data [17]. Their implementation included rotation, mirroring, and shifting, and it allowed them to perform it at zero computational cost by performing the augmentation in parallel with the training algorithm. Figure 2.1 illustrates AlexNet as a block diagram.

The community developed several architectures around AlexNet in order to learn better representations and boost generalization. Multiple contributions are listed chronologically as follows:

- **GoogLeNet** [120]. September 2014: Winner of 2014 ImageNet challenge. This model is based on the sequential concatenation of inception modules. Each module contains a stack of filters with multiple size ($k = 1, 3, 5$) to extract features at multiple scales. The inputs are padded in order to preserve dimensionality and to enable the concatenation of multi-scale features. The complexity of the network became an issue, hence, they applied a bottleneck structure using 1×1 convolutions to compress the number of feature maps and make the model computationally feasible. Figure 2.2 shows the inception module diagrams used in GoogLeNet. The architecture also contains multiple output blocks at different stages to increase the magnitude of the gradient on early layers. GoogLeNet is made of nine inception modules and three output blocks on the 3rd, 6th, and last module of the network. During testing, they adopted an aggressive cropping approach resulting in 144 crops per testing image. This testing procedure does not boost generalization but decreases false positives, which ultimately increases accuracy.

- **The All Convolutional Neural Network** [116]. April 2015. In contrast with previous models, The All CNN used small 3×3 filters and made the downsampling learnable by using strided convolutions. The classifier contains 1×1 convolutions

with average pooling and softmax activation. Removing the fully connected layers drastically reduces the parameters of the network. The novel idea was also later applied to segmentation problems [79]. They achieved competitive performance on three image classification datasets. In addition, the hidden representations can easily be visualized by inverting the data flow of the CNN. This is possible due to the fact that the model is fully convolutional.

- **Visual geometry Group Network** [114]. (VGG) April 2015. Similar to the All CNN, the VGG Net applies 3×3 convolutions but uses MaxPooling as downsampling operator. Simonyan and Zisserman [114] present several models with 11 to 19 layers. The classification structure was taken from AlexNet, which contains two fully connected layers with 4096 neurons each. In their augmentation procedure they also used random cropping which is extensively used nowadays [45, 51]. Previous architectures contain 2-3 downsampling operators, meaning that the network holds 2-3 spatial dimensions. Therefore, one of the main achievements of the VGG nets was training deep CNNs with five resolutions. Thus, enabling the network to generate more scale invariant features, yet again confirming the importance of hierarchical depth to learn visual representations. The complexity of the network is greater than previous nets due to its number of parameters. Although, it requires fewer iterations to converge due to implicit regularization given by small filters. Figure 2.3 summarizes the VGG network structure.

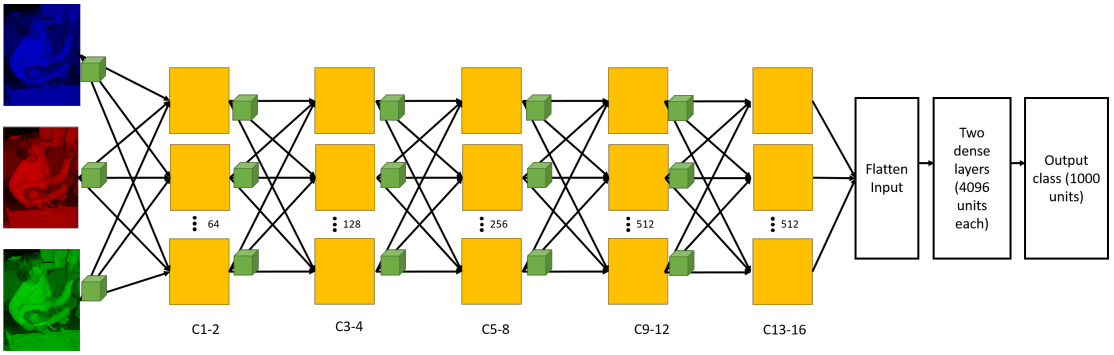

FIGURE 2.3: VGG-19 CNN applied to ImageNet with 16 convolutional and two fully connected layers, input image RGB 224x224 and outputs the class probabilities. C1-16 are convolutional layers. After every block of convolutional layers, there is a MaxPooling layer. The dimensionaly of the filters are: C1-2 64x11x11 (channels, rows, and columns of filter), C3-4 128x3x3, C5-8 256x3x3, C9-12 512x3x3, and C13-16 512x3x3. The activation of all the hidden neurons is ReLu, and of the output layer softmax.

- **Residual networks** [45, 47] (ResNets) Dec 2015. These models take benefits from residual connections, which adds skip connections that propagates the features from the previous hidden layer to the current hidden layer. In Figure 2.4 this residual connection is represented as the function *id*. The community has widely used residual connections to learn better representations at early stages of the networks. ResNets have yielded state-of-the-art results on multiple visual tasks.

These models directly alleviate the vanishing gradient problem by appending a skip connection between the input and outputs of the residual blocks. The network is built by sequential concatenation of residual blocks (ResBlocks) instead of convolutional layers. Each (original) ResBlock contains two convolutional layers, batch normalization, ReLU activation, and most importantly an identity mapping layer to merge the previous features to the output of the ResBlock. Figure 2.4 illustrates the ResBlock diagram presented in the original paper. Later studies showed empirical results using multiple ResBlocks, such as dropout and pre-activation ResBlocks. Their models contain from 52 to 1000+ layers and won the ImageNet competition 2015. In order to train deeper networks, they adopted the protocols of GoogLeNet to reduce the number feature maps, thus allowing them to train networks as deep as 1000+ layers. However, the performance decreased when using very deep models, which suggests that the vanishing gradient problem is not actually solved. In addition, Veit et al. [125] suggests that ResNets are simply an ensemble of shallower networks, which then raises uncertainty on the actual need for depth in ResNets. An additional procedure to decrease the number of parameters was changing the classification layers to follow The All CNN structure (global average pooling and softmax). The majority of recently developed architectures utilise residual connections.

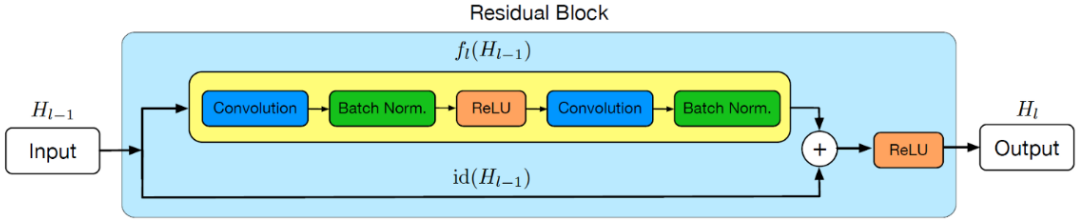

FIGURE 2.4: Residual block diagram presented in [45]. It contains two convolutional layers, batch normalization, and activation functions. The input is propagated using the identity mapping function *id*. The figure was adapted from the original paper [52].

- **Stochastic Depth Networks** [52]. March 2016. The limits of depth were pushed by using the stochastic depth protocol during training. Huang et al. [52] successfully trained very deep ResNets (1200+ layers) using a generalized version of dropout by 'deactivating' layers instead of neurons. During training, the depth of the network changes by randomly deactivating several ResBlocks. For testing and inference, the inputs are evaluated using the whole network. In theory, applying this procedure should further alleviate the vanishing gradient problem given that the magnitude of the gradient remains unchanged on deactivated layers.

- **Fractal Networks** [70] May 2017. Larsson et al. [70] present a model with multiple paths or branches to bypass one or more ResBlocks. It uses a simple expansion rule to increase the depth of the network. The architecture holds a structural

layout which are precisely truncated fractals. During training, a FractalNet randomly changes its topology and connections, which has some similarities with the Stochastic Depth procedure [52]. It is important to mention that when randomly disabling connections, there has to exist at least one path that connects input with output layer. These models benefit from "drop-path" regularization, which deactivates certain paths of the network during training. They tested FractalNets for image classification on three datasets and obtained results comparable with the state-of-the-art. In FractalNets time complexity is proportional to depth, however, going deeper does not impair accuracy.

- **Densely Connected Networks** [51] July 2017. Using ResNets as foundation, this architecture includes more than one residual connection. In fact, it appends direct connections between any two layers within the same resolution (same size of feature map). Instead of adding the previous input, this model reuses features from early layers by concatenating them on later layers. Applying this successfully leads to an increase on performance in comparisons with standard ResNets. Ultimately, the concatenation of feature maps increases the complexity of the network by a high factor, which is mainly dependent on the number of layers per resolution rather than raw depth.

Most CNNs architectures are tested on image classification problems on benchmark datasets. However, these networks have been successfully applied to other computer vision problems. Some applications include: biometrics, such as face identification [122], which can now be performed on smartphones with a high accuracy; semantic segmentation [137], for example, segmenting objects or roads, which has helped to develop self-driving vehicles; medical imaging analysis [62, 130, 88, 24], in order to assist specialists in the read of scans.

Dilated CNNs. Traditional Convolutional layers apply the operation on adjacent pixels. This enables the network to find spatial patterns with a fixed size. In order to extract bigger spatial patterns, the networks tend to downsample the images. We defined the convolution operator (using squared filters) in section 2.2 as $y_{ij} = \sum_{a=0}^{k}\sum_{b=0}^{k} w_{ab}^{t} x_{(i+a)(j+b)}$, adding dilation increases the steps a and b by the dilation factor (d). The dilated convolutional operation then becomes $y_{ij} = \sum_{a=0}^{k}\sum_{b=0}^{k} w_{ab}^{t} x_{(i+a*d)(j+b*d)}$. For example, applying a convolution with kernel size of $k \times k$ on the original image will find patterns with scale $k \times k$, however, applying the same operation on the downsampled image (downsample by 2) would extract patterns at $2k \times 2k$. This is useful to extract information at multiple scales, but there is a decay of information when applying this operation due to compression loss.

In order to overcome this, the community tends to use Dilated Convolutions. Figure 2.5 compares dilated and non dilated architectures, and shows the dimensionality of the features at every stage. It essentially contains the same amount of parameters as the

traditional convolution, with the difference that the kernel uses a receptive field with gaps accordingly to the size of the dilation. Increasing the dilation enables the network to see 'bigger' receptive fields and avoids the need to downsample the image. This, however, generates an increase in the memory required for training since the spatial size of the features are the same as the original image across the entire network. Yu et al. [133] explores the effectiveness of dilated CNNs for the tasks of object localization, image classification, and semantic segmentation. They directly show that dilated networks can outperform its non-dilated counterparts on specific tasks where spatial information is crucial. Such idea has been successfully applied in medical image segmentation [130, 88, 24].

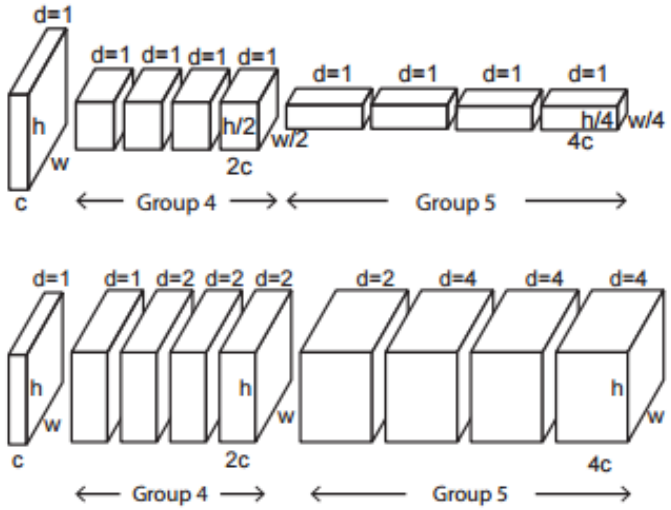

FIGURE 2.5: Comparison between CNNs, dilated (bottom) and traditional (top). d represents degree of dilation, c number of initial channels, w and h width and height respectively. Commonly used approach downsamples the features after several convolutions. Spatial size is preserved for the Dilated CNN. Figure extracted from the Dilated Residual Networks paper [133].

2.2.1 Transfer Learning

When data collection or availability becomes a problem, transfer learning can allow us to boost the performance by preliminary learning a feature extractor from a similar problem domain, often referred to as base or source dataset (large dataset). The dataset to apply transfer learning to is often addressed as target dataset (smaller dataset). Torrey and Shavlik [124] discuss several transfer learning techniques without using deep networks. These methods can be listed as follows:

Inductive transfer. The technique involves adjusting the inductive bias of the model trained on the base dataset. Thus, narrowing the hypothesis space, limiting the possible models, or remove search steps when transferring the model. In addition, inductive transfer also covers transferring Bayesian models. Multiple approaches have been explored

to transfer Bayesian learning methods. The traditional method involves computing the prior distribution on the source dataset, to then be considered in the Bayesian model developed for the target dataset [22, 85]. Another inductive transfer method includes hierarchical learning. In this case, models from multiple simple tasks are combined to generalize better in a more complex task. For example, Taylor et al. [123] propose a hierarchical learning based on tasks difficulty. Each task is scored and sorted based on difficulty. The model then learns 'easy' tasks first to obtain better results in the most complex task.

Transfer in Reinforcement Learning. In this case, an agent is in charged of performing actions given the state of the environment [115, 135]. These actions can alter the current state and obtain rewards. The global idea involves learning a policy to perform actions and maximize the cumulative reward. These models may require several iterations before learning a reasonable policy; hence, transfer learning in this setting is applied to mostly decrease the number of iterations required. These algorithms can be divided into: starting point methods, which involve training a model using a base dataset and using it as starting point for the target task; imitation methods, which allow the target model to use the policy learnt from the source dataset, which ultimately affects the updates on the target policy; hierarchical methods, which use the tasks on the base dataset as a building block for learning the target's policy.

In Deep Learning, transfer learning is applied by learning a deep network on the base dataset to then reuse the feature extractor as initialiser to train on the target dataset. For example, a common approach to transfer learning involves learning a deep network from simulated (generated) data before training on fixed real data [112]. This enables faster convergence and better generalization, and as we know simulated data is labelled for free. In the past, the idea of transferring the feature extractor using unsupervised learning methods was explored, for example, sharing a bag of visual words [12, 99]. It is important to mention that in all transfer learning tasks, the base dataset has to have similarities with the target dataset, otherwise transferring features will not boost the learning of the target dataset [65].

Several problems in computer vision are constrained by the amount of data. In order to tackle this issue, the community has adopted a transfer learning procedure to first learn a discriminative representation from a large scale dataset [103, 83, 62] . The methodology for transfer learning is divided into two stages, the first stage uses a base or source dataset and the second stage the dataset of interest or target dataset. Firstly, any CNN architecture is trained on the base dataset, typically ImageNet due to its sparsity (given the amount of labels) and size. Generalizing on ImageNet yields to a robust feature extractor, thus, features that can generalize to 1000 classes classification problem can potentially be used to generalize in other datasets with similar properties. The second stage propagates the images through the pre-trained feature extractor in order to generate robust feature vectors for the target dataset. The features can then be classified

using any machine learning algorithm. Commonly, MLPs are selected as classifier to make the network differentiable and allow fine-tuning of the complete network after training the classifier.

Given both base and target datasets, transfer learning problems can be split in two scenarios. The first problem includes having different inputs, often just a different dimensionality (it could be the case where input images are fundamentally different), which is tackled by resizing and cropping. Secondly, targets are different in both datasets. In Deep Learning for computer vision, both scenarios are typically presented; hence labels and feature space are different. The problem is then summarized to empirically quantify to which extend both statements are true. This is done by executing the transfer learning algorithm and measuring generalization on the target dataset.

Kim et al. [62] use a "bridge" dataset to co-adapt the pre-trained network to a similar space as the target dataset. For example, training on ImageNet to then fine-tune on X-Ray images, to then transfer the features to prostate cancer detection using X-Rays imaging. Yosinski et al. [132] explores transfer learning in depth by analysing the features at multiple stages of the feature extractor. Transfer learning has also been applied as an encoder decoder structure, where the pre-trained network is fixed as starting encoder.

Besides image classification from a large scale dataset to a smaller target dataset, the community has used pre-trained networks for other useful applications. One widely explored application has been semantic segmentation, which uses a pre-trained networks to compute representative feature maps of the images to segment [137, 103, 101]. These segmentation networks typically are divided into two stages; an encoding part to capture representative information from the input image (transferred representations); and, a decoding stage to target the segmentation mask. Moreover, image style transfer techniques aim to use sparse representations to capture the semantics of a "style" image and transfer it to a content image, thus generating an image that combines both content and style [35, 34]. In biometrics, transferred representations can be used as starting points to learn further more discriminative soft-biometric feature vectors [84].

These methods fpr transfering features are also applicable in other deep learning fields. For example, in Natural Language Processing, transfer learning has been applied to create a multi-lingual translation machine [58]. This methodology takes advantage of datasets with big corpora (e.g English documents) to learn better representations for a different language with less data.

2.2.2 Metrics performance

In computer vision, the type of metric is dependent on the problem to be tackled and the dataset. For classification tasks the community typically uses top-1 and top-5 accuracy, where top-1 compares ground truth with most likely class prediction; and top-5 compares

against the five predictions with the highest probability. When considering imbalance datasets, typically two methodologies are applied to measure performance. The first approach is to use weighted F_1 score, which considers the class imbalanced. This metric is further explored in Chapter 6. The second approach includes using a mean-per-class metric, where predictions are normalized accordingly to the number of samples per class. If the imbalance of the dataset is not considered, the results can be misleading and a biased or overfitted network might be overlooked.

Besides these metrics, we often utilise the loss value (of both training and validation set) as training indicator. Again, the loss function to use depends entirely on the nature of the problem. For classification tasks, cross-entropy or mean squared error are often the functions of choice [67, 114, 45]. For object localization, mean Average Precision (mAP) has been widely applied to evaluate models. It computes the average of the precisions at different recall values, in other words, it considers the predicted position of objects and the overlapping area to determine whether the object was detected correctly. In addition, Lin et al. [76] present Focal Loss, which aims to learn more from hard examples and avoids the model from overfitting on easy negatives. State-of-the-art image generation is evaluated using an inception model (GoogLeNet) to do inference on the generated images [7]. If the generated image fools GoogLeNet, an accumulator is increased; thus, providing accuracy numbers based on how many times GoogLeNet was fooled by the generator.

The loss function is selected depending on the nature of the targets. In multi-label classification tasks the community often uses Cross-Entropy loss. This function can be expressed as $-\sum_{c=1}^{L} y_{o,c} \log(\hat{y}_{o,c})$, where c is the class index, L the number of classes, o the observation index, y represents the target, and $\hat{y}$ the prediction. Using this loss function for classification tasks has shown speed ups in comparison to other losses, such as Mean Squared Error.

2.2.3 Datasets

State-of-the-art models are developed and tested on benchmark datasets. The following datasets are widely used by the computer vision community to test classification models:

2.2.3.1 MNIST

MNIST is a well known hand written digit recognition dataset [72]. The data is splitted into 60k images for the training set, and 5k images for the test set. The images are binary (one channel per image), with a size of 28×28. Typically, this dataset is easy to learn and it is used to generate early results. The labels are balanced and labelled from zero to nine. A random subset of images from MNIST can be is shown on the left of Figure 2.6.

FIGURE 2.6: Nine samples from two datasets. Left) MNIST Right) CIFAR. The images were resized for proper visualization.

2.2.3.2 CIFAR-10/100

Tiny image classification dataset containing 60k images [66]. There are three datasets extracted from CIFAR, they contain similar images and targets are 10, 20 or 100 labels. These targets describe different objects, such as animals and vehicles. The input images are RGB of size 32×32. Figure 2.6 shows random input samples from the CIFAR-10 dataset. In comparison with MNIST, this dataset is more complex given that the labels are more sparse and less representative of the label.

2.2.3.3 ImageNet

Large scale image classification dataset containing 1.3M images for training and 0.1M for testing [109]. In contrast with previously visited datasets, ImageNet contains RGB images with multiple size from a range between 100 and 500 pixels per dimension. This dataset is widely used in transfer learning given that it contains 1000 classes and models can learn very robust representations. By generalizing on ImageNet, the network can learn to generalize on similar domains to image classification. The metrics used are top-5 and top-1 accuracy to quantify performance on this dataset. Human performance on this dataset oscillates around $5 \pm 1\%$ top-5 accuracy [46]. A random set of images selected from ImageNet are shown in Figure 2.7. Since the ImageNet dataset was released, deep networks have taken over the state-of-the-art. Figure 2.8 shows the performance of deep architectures chronologically.

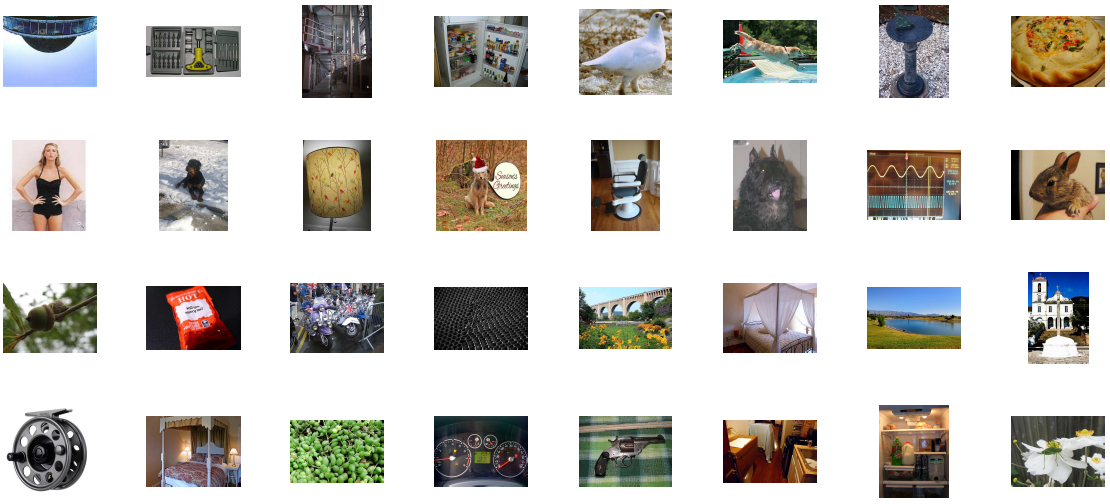

FIGURE 2.7: Thirty-two random samples from the ImageNet dataset. The images were resized for illustration purposes. Note that the images do not have the same size.

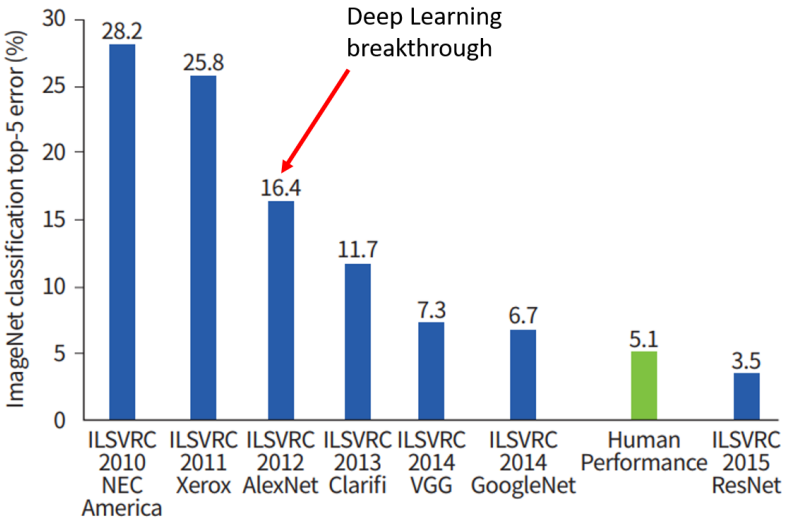

FIGURE 2.8: Results for the ImageNet large scale object detection competition on chronological order. Deep Learning was first applied on 2012. The error has drastically decreased since then given improvements on Deep Learning architectures.

2.3 Limitations of Deep Learning in Computer Vision

Even though deep learning has generated outstanding performance in computer vision tasks, training networks can be a complex and inflexible procedure. Some drawbacks of deep learning are listed as follows:

- **Time complexity**. Training on the required amount of data can be computationally expensive due to the cost of propagating high dimensional inputs through the network.

- **Space complexity**. The number of parameters of the network increases with depth. Hence, the depth of the network is limited by the hardware. Compressing, pruning, and binarizing the weights have been proposed to decrease the number of parameters of the network [100, 42]. In addition, training these models requires to store the feature maps at every stage of the network, which can be very computational expensive when the images are high resolution.

- **Vanishing gradient problem**. Explored in detail in Chapter 4, it affects the learning capabilities of early layers. Ideally, one would expect that by solving the vanishing gradient problem, the performance should remain stable when appending new layers to the architecture.

- **Appending new targets**. For some applications, it is imperative to preserve the robustness of the network while learning a new label. This is particularly useful in areas like biometrics, on which new subjects may be appended to the database at any point. A commonly used approach involves using weak flexible algorithms (e.g. KNN) with DL pre-trained features [122]. This would allow to use the flexibility (of adding new labels) of the algorithm with robust deep representations.

- **Data size**. Networks tend to overfit when insufficient data is provided, this applies to any deep network. Transfer Learning, as described previously in this chapter, it is a solution that works in some cases when the base and target datasets have a similar structure. Traditional approaches still outperform deep learning when the amount of data is limited, thus, making deep learning inapplicable. In some cases augmenting the data can help to generate more variants of the training dataset and simulate an increase in the data size [67].

2.4 Deep Learning for Signal Processing

Multiple studies related to time series and speech have taken over the state-of-the-art using deep learning. Signals have similar properties as images which enables the community to share DL architectures between image and signal processing. Moreover, the convolutional operator in computer vision includes a spatial component, while in this section we elaborate around convolutions in the temporal domain (1D convolutions). The filters are optimized to learn patterns between adjacent time components as opposed to spatial patterns.

ResNets and its derivatives have an equivalent architecture in the temporal domain [30]. However, there has been recent improvements with WaveNet [64] style nets, which include dilated temporal convolutions and multi-activation layers. The dilated convolutional operator in signal processing is equivalent to performing a convolution at a lower sampling frequency, allowing the network to find longer term patterns with the same number of parameters. The dilated convolution operation in this setting is further discussed in Chapter 6. Kingma et al. [64] applies deep dilated networks with double activation on multiple speech problems, such as speech recognition, text-to-speech, and speech enhancement.

Traditional approaches to solve signal processing problems include classifying bespoke features, such as Mel-frequency Cepstral Coefficients (MFCC), statistical features (median, mean, area under the signal), often including derivatives of these features. In addition, traditional machine learning algorithms, including Support Vector Machines (SVMs) and Random Forest, have been applied to separate these handcrafted features [56]. In Chapter 6 we explore a Human Activity Recognition (HAR) problem. HAR uses multi-sensor data that can be addressed as time series classification [138].

Networks in signal processing also make use of the convolutional operator, however it differs in dimensionality with the operation applicable to computer vision. The convolution is unidimensional and aims to extract temporal patterns. For this reason it is frequently referred as temporal convolution. Figure 2.9 illustrates the temporal convolution operator, the filters C_j are convolved with the input series to generate F feature maps (feature signals).

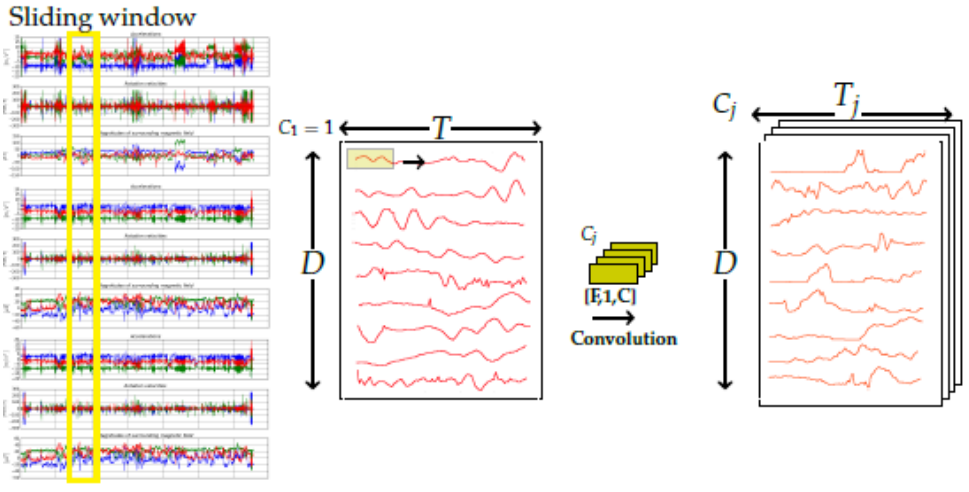

FIGURE 2.9: Temporal convolution with D signals, window size of T, and F features. The figure was adapted from [89].

2.5 Time Series Classification

In this task the aim is to predict a label given some time dependent variables. For example in forecasting, the goal is to successfully predict when an event is going to happen given historical temporal data. Before Deep Learning, the state-of-the-art in time series classification involved nearest neighbour (k-NN) and Dynamic Time Warping (DTW) [8]. They key idea was generating invariant features which would be dependent on the domain, e.g. invariant to warping. These robust features were robust enough to be successfully classified by a simple KNN algorithm with a domain specific distant measurement [8].

State-of-the-art models use feature learning instead of handcrafted domain specific features. Wang et al. [127] perform preliminary tests on MLPs, and CNNs on 44 benchmark datasets, and concluded that both architectures can provide competitive results. We believe CNNs should outperform MLPs in time series analysis, given that MLPs are not desing to extract discriminative features across the time axis. Moreover, the receptive field of CNNs is very limited, hence, dilated kernels enables them to find long term dependencies that can potentially be weakly captured by the MLP. At this point, LSTMs have shown better performance for this classification task [38].

Recently MLPs and LSTMs have been shown to be outperformed by Deep CNNs on for time series classification [6]. Borovykh et al. [10] explore more complex CNN structures and show state-of-the-art performance on predicting sequences.

2.6 Summary

In this chapter we discussed the literature necessary to fully comprehend the next chapters. We discuss early approaches using neural networks as well as the training methodology carried out to learn meaningful representations. The chapter starts exploring the single unit neural networks (perceptron), and further elaborate on state-of-the-art deep models. In Section 2.1 we list, explain, and revise important deep learning concepts. We then discuss Deep Learning in the computer vision setting, thus, revising modern models and architectures applied on image classification tasks. Figure 2.8 illustrates chronologically the performance of these state-of-the-art models. In addition, we work around the limitations of deep learning, such as data availability, and some approaches to overcome these limitations (e.g Transfer Learning). We briefly explore deep learning for signal processing. Specifically, we review time series classification, which will be further discussed and applied in Chapter 6.

Chapter 3

Layer wise training

In Chapter 2 we discussed the literature on deep learning. In addition, we covered several disadvantages of fixed end to end training. We believe layer-wise training can alleviate some of these disadvantages. Improvements are obtained as a consequence of directly tackling the vanishing gradient problem, as well as better adaptability due to the growing structure. Chapter 4 covers our layer by layer training algorithm which generates complexity improvements of the network.

In this chapter we explore the literature on layer by layer training of neural networks, from early approaches on stacking perceptrons, to more recent developments in image generation.

3.1 Cascade Correlation

Fahlman and Lebiere [28] present an algorithm to iteratively train network of perceptrons. This cascade architecture appends one unit (perceptron) per iteration. On each iteration a new unit is learnt, while previously learnt units remain frozen. Once the unit is appended to the system, the algorithm aims to maximize the magnitude of the correlation between the most recent unit's output and the residual error signal of the network (see S in Equation 3.1). Considering Equation 3.1, Cascade Correlation aims to maximize S given the error signal E_o, the features V_p (also referred to as patterns). $\bar{E_o}$ and $\bar{V}$ represent the average over all units.

$$S = \sum_{o} | \sum_{p} (V_p - \bar{V})(E_{p,o} - \bar{E_o}) | \tag{3.1}$$

Details of the algorithm may vary depending on the application. In the particular case of study, Fahlman and Lebiere [28] applied Cascade Correlation to a two spiral problem,

which consists of 194 inputs with binary labels. The algorithm then can be written as the following sequence of steps:

- The first iteration starts with a single neuron network. This small network is trained until the error function plateaus
- Create several candidate units, and evaluate the performance of the network given these candidates.
- Select the candidate with the maximum covariance between the unit's output E_o and the features already learnt V. Once the candidate unit is selected, it is appended to the network and its parameters are frozen.
- Repeat the process of appending new units until the stopping criteria is reached (e.g maximum number of iterations, or validation error cannot be improved).

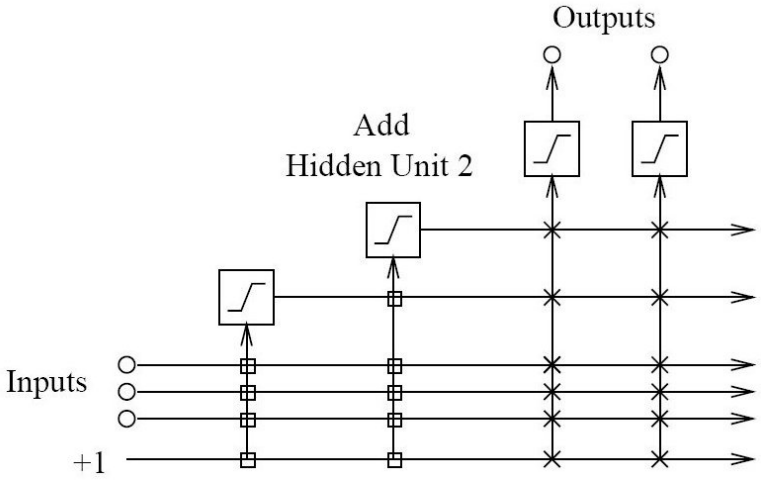

FIGURE 3.1: Cascade Correlation Diagram. One unit has already been trained, boxes connections are frozen, X connections belong to the current iteration and are trainable. Figure adapted from the original paper [28]

This idea was one of the first approaches to "layer by layer" training. The units are densely connected, which can be seen as appending residual connections between the newly appended unit and previously learnt features. Unravelling a ResNet generates a similar structure as Figure 3.1, which suggests the Cascade Correlation makes use of residual connections during training [125]. In addition, Fahlman and Lebiere [28] boosted Cascade Correlation's performance by applying an early version of dropout to deactivate frozen (already learnt) units during training, except that the already learnt perceptrons are fixed, hence the vanishing gradient is irrelevant. The candidate units may contain different activation functions. However, in the original paper they only use pools of units with same activation function. They do mention the possibility of mixing activation function in the network. Figure 3.1 shows the connectivity of the algorithm for a Cascade Correlation network with one frozen unit.

One of the achievements of this algorithm is to adaptively find the architecture of the network and its connectivity pattern. By applying this type of layer-wise technique, one may specify a stopping criterion to determine when to stop appending new units. This is beneficial given that it reduces the amount of empirical tuning required to find the optimal minimum in the error function. This method enables the possibility of training deeper networks with limited resources, since the features can be cached at any point in time given that the rest of the network is constant. At the end, the structure allow us to have a trade of between time and space, which can be tuned accordingly to the available resources. The Cascade Correlation algorithm was tested on the two spiral and the n-input parity problem.

Inspired by this algorithm, we developed the Cascade Learning algorithm, which aims to generalize advantages of Cascade Correlation to modern Deep Learning. This novel algorithm is further explained on Chapter 4.

3.2 Adaptive architectures

In this section we briefly discuss a couple of adaptive architectures that followed up Cascade Correlation. These models were developed prior deep learning.

3.2.1 Resource-Allocating Networks (RAN)

Platt [96] proposes RAN, which is an adaptive architecture that aims to append a new computational unit whenever a new pattern is presented in the data. It resembles a Radial Basis Function network (RBF), with the main difference that each stage is trained sequentially in a layer-wise fashion. The model starts with no hidden units and grows iteratively depending on the novelty of an observation. The output of a RAN is a linear combination of the hidden units responses. Equation 3.2 shows the mechanics to evaluate a RAN, where K is the current number of patterns, α_k for $k = 1$ to K are the weights of the layer (α_0 is the bias), and ϕ_k represents the response of the hidden unit k given the input x. The responses can be obtained by measuring the distance between the existing patters u_k and the input data (see Equation 3.3). The network decides to store a new pattern u_{K+1} depending on its novelty, which can be computed by evaluating two conditions. These conditions are dependent on thresholds ϵ_n (scale of resolution in input space) and e_{min} (minimum error to be achieved): (a) $||x_n - u_{nr}|| > \epsilon_n$, input vector has to be far away from the existing patterns; (b) $e_n = y_n - f(x_n) > e_{min}$, error between outputs and targets has to be more than the predefined error threshold. When these conditions are not met, the network uses Leas Mean Squares (LMS) [128] gradient descent to update the coefficients α_k and the stored patterns u_k and it does not append a new unit to the model. These models showed efficiency and adaptability for the time series prediction task.

$$f(x) = \alpha_0 + \sum_{k=1}^{K} \alpha_k \phi_k(x) \tag{3.2}$$

$$\phi_k(x) = \exp\Big(-\frac{1}{\sigma_k^2} ||x - u_k||^2 \Big) \tag{3.3}$$

Kadirkamanathan and Niranjan [60] propose RAN-EKF, which is an enhanced version of the original RAN. The main contribution is the application of the extended Kalman filter to update the weights instead of using LMS. The resulting network holds fewer parameters, can obtain better performance when applied to a time series prediction problem, and requires less time to converge.

3.2.2 Adaptive-Network-Based Fuzzy Inference (ANFIS)

Jang [57] proposes an adaptive architecture based on fuzzy inference that work as high-level reasoning mechanisms. It also includes a neural network component that incorporate low-level reasoning. It sequentially tunes fuzzy if-then rules stated by human expertise. These rules are tuned using the learnable units, and they describe input-output behaviour of the complex data. They directly compare their approach with Cascade Correlation [28] even though is not an incremental adaptive architecture. Instead, it starts with a fixed structure and it adapts based on a hybrid learning rule. Thus, this model progressively removes units, which can be seen as a reverse layer-wise (unit-wise) algorithm. For the purpose of this thesis, fuzzy logic is not considered. State-of-the-art deep networks are more alike the networks generated by the Cascade Correlation algorithm.

3.3 Deep Belief Networks (DBNs)

Hinton et al. [48] introduced a methodology to train deep multi-layer perceptrons. The algorithm can be summarized in two stages: the first stage of unsupervised adaptation of the weights using Restricted Boltzmann Machines (RBMs) [2] also referred in the literature as autoencoders; and a second stage to fine-tuning in a supervised fashion to co-adapt the features to the target class. The algorithm does not include an adaptation criterion to determine how many layers to use. Instead, it requires the structure of the network to be known before executing the algorithm.

Unsupervised pre-training. In this first stage, the algorithm initialises one hidden layer h_1 and aims to model the raw input to satisfying $x_i = h_i^{(1)}$. For the next stage, $h_i^{(1)}$ is considered the visible layer and its parameters are frozen until the last stage of the algorithm. For the next layer the "new data" is gathered by propagating the inputs

through the frozen layer, thus obtaining a new input matrix $h_i^{(1)}$, hence the pseudo-inputs would consist of features that approximations of the input data. Then the next layer is trained using the same methodology to obtain a layer that satisfies $h_i^{(2)} = h_i^{(1)}$. The stacking process is iteratively repeated until reaching the desired number of layers. Each iteration can be seen as learning one autoencoder.

Fine-tuning. Once all the layers have been pre-trained, all the parameters are enabled to minimize the error function between inputs and targets on the bottom of the network. The minimization of the negative log likelihood is performed using supervised gradient descent. This stage is often regularized using dropout and weight decay.

This algorithm enabled the training of deep networks without massively overfitting the training data, and allowed neural networks to catch up with other machine learning approaches. Bengio et al. [9] further explores these networks using empirical analysis. They concluded that a greedy layer wise training mostly helps optimization by initialising weights in a region near a good local minimum. In addition, they briefly explored a supervised greedy layer wise algorithm, however, their conclusion suggests that a semi-supervise layer-wise strategy can yield better performance of the overall DBN.

3.3.1 Convolutional DBNs (CDBN)

Similarly, Lee et al. [73] present an algorithm to sequentially pre-train CNNs using convolutional RBMs. Once the feature extractor is trained, it computes an SVM to classify these features. They showed how the features were learnt hierarchically and generate more complex feature extractor on deep layers. In addition, they demonstrate the effectiveness of max pooling to downsample the feature maps. Even though this approach is mostly unsupervised, Convolutional DBNs (CDBN) perform better when there is more labelled data, and yields competitive results when the data is limited.

3.4 Layer-wise training using kernel similarity

Kulkarni and Karande [68] present an algorithm to sequentially train layers of perceptrons. The algorithm aims to maximize the distance between features of different classes, and minimize the distance between features belonging to the same class. On every iteration there is a new set of inputs propagated to the next layer defined as $X_k = tanh(D_{k-1}W_k)$, where X_k is the feature matrix at stage k, and W_k the weights of the same stage. Before every gradient update, the algorithm also normalizes the features to have zero mean and unit l_2 norm. They showed comparable performance with traditional end to end training of MLPs on MNIST and CIFAR-10 datasets. In addition, they performed a kernel analysis of these networks and showed better feature encoding

than early approaches. Given that they train a layer at the time, this algorithm is less computationally demanding than the alternatives.

3.5 AdaNet

Adaptive Structural Learning (AdaNet) progressively grows the size of the network. Cortes et al. [20] explore this idea for Deep MLPs, and show a theoretical proof of convergence. The algorithm starts by randomly generating several candidate structures (subnetworks; not necessarily single layers), training such candidates using traditional Stochastic Gradient Descent (SGD), and evaluating the given candidates. It then selects the candidate that produces the smallest loss, appends it to the network and discards the others. For the remaining iterations it performs a similar procedure: randomizing candidates, training, evaluating, selecting. As opposed to DBNs, the connectivity in this case is also randomized. It does not necessarily stack the subnetworks at the last stage of the current network, instead, it can be the case where the randomized candidate connects side by side with the existing units. The algorithm continues until the last iteration has been reached or none of the candidates contribute to decreasing the loss. This pool of subnetworks idea was explored first in the Cascade Correlation algorithm [28]. One of the main advantages is the easy parallelisation of the training of subnetworks. Algorithm 3.5 summarizes these steps in pseudocode. The algorithm was compared with traditional training of MLPs on paired subsets of CIFAR-10 (several binary classification problems). The learnt topology of the network using AdaNet provides better performance than those trained using grid search. Moreover, using grid search to find the 'ideal' topology of the network can be computationally demanding. Finally, this idea can be generalized to more complex architectures, such as CNNs and RNNs.

3.6 Progressive Generative Adversarial Networks (PGGANs)

Most recently, Karras et al. [61] trains generative model for super-resolution image generation. PGGAN starts with tiny images and progressively stack convolutional layers to both the generator (network in charge of generating the image) and discriminator (network to evaluate the quality of the generated image) to smoothly adapt the network to a higher resolution. In early resolutions, the generation of images is more stable due to less class information and fewer modes [93]. Figure 3.2 shows the progressive growth of the model, starting with a resolution of 8×8, and converging to a resolution of 1024×1024. In other words, the model first learns large-scale structure (e.g context) to then focus the attention on a greater level of detail. Training early iterations (low resolution generation) is a simpler problem than training at full resolution (e.g 1024×1024). This was demonstrated given that the quality of the generated images decreases with smaller

Algorithm 1 Pseudocode of AdaNet with two candidates. f_t represents the network at step t, w parameters of the network, F_t loss of the network, and h_t selected candidate at step t. The pseudocode was adapted from the original paper [20].

procedure AdaNet(X, Y, T) ▷ input data and number of iterations
  $f_0 = 0$ ▷ Initialise model
  for $t = 0 : T$ **do** ▷ Iterate from 0 to T
    $h, h' \leftarrow GetRandomSubnetworks(f_{t_1})$ ▷ Initialise candidates
    $w \leftarrow minimize(F_t(w, h))$ ▷ SGD with first candidate
    $w' \leftarrow minimize(F_t(w, h'))$ ▷ SGD with second candidate
    if $F_t(w, h) \leq F_t(w', h')$ **then** ▷ Select candidate
      $h_t \leftarrow h$
    else
      $h_t \leftarrow h'$
    end if
    if $F(w_{t-1}) + w^* < F(w_{t-1})$ **then** ▷ Update net or stop if no improvement
      $f_{t-1} \leftarrow f_t + w^* h_t$
    else
      return f_{t-1}
    end if
  end for
  return f_T
end procedure

resolutions [93]. This layer wise training also provides time complexity advantages of up to 2-6 times, since most iterations are executed at low resolution. As a result, the algorithm is capable of generating images at 1024^2 resolution. They also proposed a methodology to enable better convergence of the network by minimizing the conflict between generator and discriminator. As opposed to the other layer-wise algorithms, a progressive GAN grows the depth of both encoder and decoder of the generator, which can be seen as the network progressively getting deeper at a higher rate.

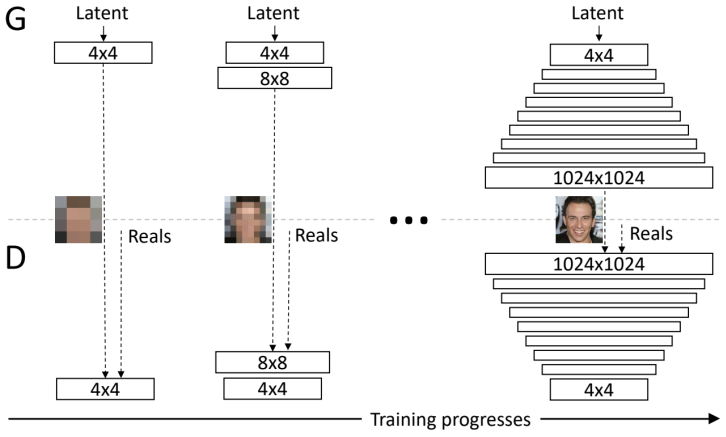

Figure 3.2: Diagram illustrating the progressive growth of GANs. Both generator and discriminator grow and duplicates resolution with each iteration. It starts with small 8×8 resolution, and finishes with 1024×1024. Image adapted from the original paper [61].

The PGGANs were tested on CELEBA-HQ dataset, which is a high resolution version of CELEBA [77]. This dataset was developed due to the size of the spatial dimensions of the CELEBA datasets, and to fix inconsistencies that would induce noise into the network, such as images containing more than one face. Additionally, the images were modified to improve the overall quality. These modifications include a convolutional autoencoder to remove JPEG artifacts in images [81], and an adversarial network trained to increase the resolution of the image by 4. The faces are then cropped from the enhanced images. This process was applied on all 202599 images. The dataset generation procedure also included quantifying the image quality for the whole dataset and selecting the best 3000.

3.7 Summary

Through the chapter, we covered the literature on layer by layer training. Layer by layer training has been applied and studied in the past. The main advantages of such training may include memory and time complexity (e.g Cascade Correlation, AdaNet), and performance improvements (e.g DBNs, PGGANs). In addition, these layer-wise models often include adaptive hyperparameters, which reduces the effect of poorly choosing these hyperparameters. The Chapter starts from early approaches applied to perceptrons, and follows by more complex generative architectures. Some approaches include unsupervised learning, such as RBMs and autoencoders. In the following chapter we present our own supervised layer-wise algorithm with similar advantageous properties.

Chapter 4

Cascade Learning Architecture for Deep Convolutional Neural Networks

In this chapter, we propose a novel approach for efficient training of deep neural networks in a bottom-up fashion using a layered structure. Our algorithm, which we refer to as Deep Cascade Learning, is motivated by the Cascade Correlation approach of Fahlman and Lebiere [28] who introduced it in the context of perceptrons and is further explained in Chapter 3. We demonstrate our algorithm on networks of convolutional layers, though its applicability is more general and can potentially be extended to other feedforward architectures. In later chapters we evaluate this framework on two more settings, a multi-sensor data problem, and in transfer learning.

Training of deep networks in a cascade directly circumvents the well-known vanishing gradient problem by ensuring that the output is always close to the layer being trained. We present empirical evaluations comparing our deep cascade training with standard end to end training using back propagation of two convolutional neural network architectures on benchmark image classification tasks (CIFAR-10 and CIFAR-100). In addition, we show that our intuitions about gradient magnitudes are correct and then investigate the features learned by the approach. We find that better, domain-specific, feature representations are learned in early layers when compared to what is learned in end to end training. Domain specific features at early layers might no be necessary to obtain good generalisation of the overall model, however, this is useful for tasks were early representations are used or transferred. The increase in robustness of the features at early stages is partially attributable to the vanishing gradient problem which inhibits early layer filters from changing significantly from their initial values. While both networks perform similarly overall, recognition accuracy increases progressively with each added layer, with discriminative features learnt in every stage of the network, whereas

in end to end training, no such systematic feature representation was observed. We also show that such cascade training has significant computational and memory advantages over end to end training, and can be used as a pre-training algorithm to obtain better performance.

The reminder of the chapter is organized as follows. Section 4.1 explains the Cascade Learning algorithm and analyses its advantages. Section 4.2 shows the results and discussion of two experiments performed on two architectures. Finally, Section 4.3 summarizes the findings, contributions, and potential further directions.

4.1 The Deep Cascade Learning Algorithm

In this section we describe the proposed Deep Cascade Learning algorithm and discuss the computational advantages of training in a layer-wise manner. All the code used to generate the results in this manuscript can be found in the GitHub repository available at *http://github.com/EnriqueSMarquez/CascadeLearning*.

4.1.1 Algorithm description

As opposed to the cascade correlation algorithm, which sequentially trains perceptrons, in Deep Cascade Learning we cascade layers of units. The proposed algorithm allows us to train deep networks in a cascade-like, or bottom up layer-by-layer, manner. For the purposes of this Chapter, we focus on convolutional neural networks architectures. The deep cascade learning algorithm splits the network into its layers and trains each layer one by one until all the layers in the input architecture have been trained, however, if no architecture is given, one can use the cascade learning to train as many layers as desired (e.g. until the validation error stabilizes). This training procedure allows us to counter the vanishing gradient problem by forcing the network to learn features correlated with the output on each and every layer. The training procedure can be generalized as "several" single layer convolutional neural networks (sub-networks) that interconnect and can be trained one at a time from the bottom up (see Figure 4.1). The idea of incorporating more output heads is not new for the community, several layer-wise algorithms take into consideration the use of output blocks at different stages of the network. GoogLeNet [120] also makes use of multiple output heads, with the main difference that the whole network is trained in an end to end fashion.

The algorithm takes as inputs the hyper-parameters of the training algorithm (e.g. optimizer, loss, epochs) and the model to train. Pseudocode of the Cascade Learning procedure can be found in Algorithm 2, and will be referred to in further explanations of the algorithm. Learning starts by taking the first layer of the model and connecting it to the output with an 'output block' (line 9), which might be several dense layers

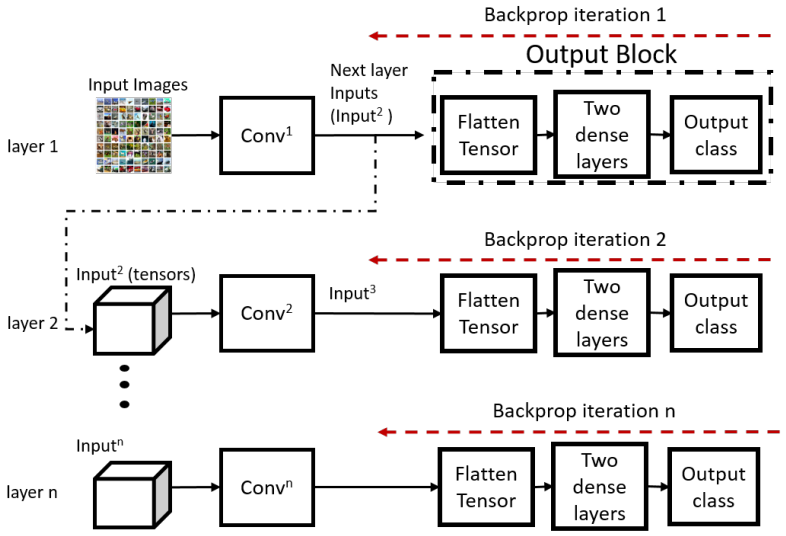

FIGURE 4.1: Overview of Deep Cascade Learning on a convolutional network with n layers. Input^i is the tensor generated by propagating the images through the layers up to and including Conv^{i-1}. Training proceeds layer by layer; at each stage using convolutional layer outputs as inputs to train the next layer. The features are flattened before feeding them to the classification stage. In contrast with the cascade correlation algorithm, the output block is discarded at the end of the iteration (see Algorithm 2), and typically it contains a set of fully connected layers with non-linearities and dropout.

connected to the output [67, 114], or, as it is sometimes shown in the literature, an average pooling layer and the output layer with an activation function [116]. The output block specifications are chosen based on the task and architecture to use. Training using standard backpropagation then commences (using the pre-supplied parameters; see the loop in line 11) to learn weights for the first model layer (and the weights for the output block). Once the weights of the first layer have converged, the second layer can be learned by connecting it to an output block (with the same form as for the first layer, but potentially different dimensionality), and training it against the outputs with pseudo-inputs created by forward propagating the actual inputs through the (fixed) first layer. This process can then repeat until all layers have been learned. At each stage the pseudo-inputs are generated by propagating the actual inputs forward through all the previously trained layers. It should be noted that once a layers' weights have been learned, they are fixed for the training of all subsequent layers. Figure 4.1 gives a graphical overview of the entire process.

Most hyper-parameters in the algorithm remain the same across each layer, however we have found it beneficial to dynamically increase the number of learning epochs as we get deeper into the network. Additionally, we start training the initial layers with orders of magnitude fewer epochs than we would if training end to end. The rationale for this is that each sub-network fits the data faster than the end to end model and we do not want to overfit the data, specially at in the lower layers. Overfitting in the lower layers

Algorithm 2 Pseudocode of Cascade Learning adapted from the Cascade Correlation algorithm [25]. Training is performed in batches, hence every epoch is performed by doing backpropagation through all the batches of the data.

procedure CASCADE LEARNING($layers, \eta, epochs, epochsUpdate, out$)
  Inputs $layers$: model layers parameters (loss function, activation, regularization, number of filters, size of filters, stride)
    η : Learning rate
    $epochs$: starting number of epochs
    k : epochs update constant
    out : output block specifications
  Output W : L layers with $\mathbf{w}_l$ trained weights per layer
  for $layer_l = 1 : L$ **do** ▷ Cascading through trainable layers
    Init $layer_l$ *and connect output block*
    $i_l \leftarrow epochs + k \times layer_l$
    for $i = 0; i{+}{+}; i < i_l$ **do** ▷ Loop through data i_l times
      $\mathbf{w}_l^{new} \leftarrow \mathbf{w}_l^{old} - \eta \nabla J(\mathbf{w}_l)$ ▷ Update weights of layer l by gradient descent
      if *Validation error plateaus* **then**
        $\eta \leftarrow \eta/10$ ▷ Change learning rate if update criteria is satisfied
      end if
    end for
    Disconnect output block and get new inputs
  end for
end procedure

would severely hamper the generalisation ability of later layers. In our experiments we have found that the number of epochs required to fit the data is dependent on the layer index, if a layer requires $i_{(epochs)}$, the subsequent layer should require $i_{(epochs)} + k$, where k is a constant whose value is set dependent on the dataset. This hyperparameter (k) can be chosen empirically by executing the algorithm in a small subset of the data.

A particular advantage of such cascaded training is that the backward propagated gradient is not diminished by hidden layers as happens in the end to end training. This is because every trainable layer is immediately adjacent to the output block. In essence, this should help the network obtain more robust representations at every layer. In Section 4.2 we demonstrate this by comparing confusion matrices at different layers of networks trained using Deep Cascade Learning and standard end to end backpropagation. The other advantage, as we demonstrate in the following subsections, is that the complexity of learning is reduced over end to end learning, both in terms of training time and memory.

4.1.2 Cascade Learning as supervised pre-training algorithm

A particular appeal of deep neural networks is pre-training the weights to obtain a better initialization, and further achieve a better minimum. Starting from the work of Hinton *et. al* on Deep Belief Networks [48], unsupervised learning has been considered

in the past as effective pre-training, initializing the weights which are then improved in a supervised learning setting. While this was a great motivation, recent architectures [46, 45, 70], however, have ignored this and focused on pure supervised learning with random initialization.

Deep Cascade Learning can be used to initialize the filters in a CNN and diminish the impact of the vanishing gradient problem. After the weights have been pre-trained using Deep Cascade Learning, the network is tuned using traditional end to end training (both stages are supervised). When applying this procedure it is imperative to re-initialize the output block after pre-training the network, otherwise the network would rapidly reach the sub-optimal minimum obtained by the cascade learning. This does not provide better performance in terms of accuracy. In later sections we discuss how this technique may lead the network to better generalization.

4.1.3 Time Complexity

In a convolutional neural network the time complexity of the convolutional layers is:

$$O\left(\sum_{l=1}^{d} n_{l-1}\ s_l^2\ n_l\ m_l^2\ i\right), \tag{4.1}$$

where i is the number of training iterations, l is the layer index, d is the number of layers, n is the number of filters, s and m is the size of the input and output (spatial size) respectively[1] [43].

Training a convolutional neural network using the Deep Cascade Learning algorithm changes the time complexity as follows:

$$O\left(\sum_{l=1}^{d} n_{l-1}\ s_l^2\ n_l\ m_l^2\ i_l\right), \tag{4.2}$$

where i_l represents the number of training iterations for the l-th layer. The main difference between the equations is the number of epochs for every layer, in Equation 4.1 i is constant while in Equation 4.2 depends on the layer index. Note in this analysis, we have purposefully ignored the cost of performing the forward passes to compute the pseudo-inputs as this is essentially 'free' if the algorithm is implemented in two threads (see below). The number of iterations in the cascade algorithm depends on the dataset and the model architecture. The algorithm proportionally increases the number of epochs on every iteration since the early layers must not be overfit, while later layers should be

[1] Note that this is the time complexity of a single forward pass; training increases this by a constant factor of about 3.

trained to more closely fit the data. In practice, as shown in the simulations (Section 4.2), one can choose each i_l such that $i_1 << i$ and $i_L \leq i$, and obtain almost equivalent performance to the end to end trained network in a much shorter period of time. If $\sum_{l=1}^{d} i_l = i$, the time complexity of both training algorithms is the same, noting that improvements coming from caching the pseudo-inputs are not considered.

There are two main ways of implementing the algorithm. The best and most efficient approach is by saving the pseudo-inputs on disk once they have been computed; in order to compute the pseudo-inputs for the next layer one only has to forward propagate the cached pseudo-inputs through a single layer. An alternate, naive, approach would be to implement the algorithm using two threads (or two GPUs), with one thread using the already trained layers to generate the pseudo-inputs on demand, and the other thread training the current layer. The disadvantage of this is that it would require the input to be forward propagated on each iteration. The first approach can further decrease the runtime of the algorithm and the memory required to train the model at the expense of disk space used for storing cached pseudo-inputs.

4.1.4 Space complexity

When considering the space complexity and memory usage of a network, we have to consider both the number of parameters of the model, and also the amount of data that needs to be in memory in order to perform training of those parameters. In standard end to end backpropagation, intermediary results (e.g. response maps from convolutional layers and vectors from dense layers) need to be stored for an iteration of backpropagation. With modern hardware and optimisers (based on variants of mini-batch stochastic gradient descent) we usually consider batches of data being used for the training, so the amount of intermediary data at each layer is multiplied by the batch size.

Aside from offline storage for caching pseudo-inputs and storing trained weights, the cascade algorithm only requires that the weights of a single model layer, the output block weights, and the pseudo-inputs of the current training batch are stored in RAM (on the CPU or GPU) at any one time. This *potentially* allows memory to be used much more effectively and allows models to be trained whose weights exceed the amount of available memory, however this is drastically affected by the choice of output block architecture, and also the depth and overall architecture of the network in question.

To explore this further, consider the parameter and data complexity of a VGG-style [114] network of different depths (this model is explained in detail in Chapter 2. Assume that we can grow the depth in the same way as going between the VGG-16 and VGG-19 models in the original paper by [114] (note we are considering Model D in the original paper to be VGG-16 and Model E to be VGG-19), whereby to generate the next deeper architecture we add an additional convolutional layer to the last three blocks of similarly

model	parameters	data storage	total
VGG-16	13.9M	32K	14.2M
VGG-19	19.1M	338K	19.5M
VGG-22	24.5M	364K	24.8M
VGG-25	29.8M	391K	30.2M
VGG-28	35.1M	418K	35.5M

TABLE 4.1: Storage required of end to end training of various depths of VGG style networks. The number of parameters increases with depth. The data storage units of the training depends on the computational precision.

trainable layer #	parameters	data storage	total
1	33.6M	69K	33.7M
2	8.6M	132K	8.6M
Pooling			
3	16.9M	50K	17.0M
4	4.5M	66K	4.6M
Pooling			
5	8.8M	25K	8.8M
6	2.8M	25K	2.9M
...			

TABLE 4.2: Storage required of cascade training of various layers of a VGG style network. The number of parameters decreases with depth. The data storage units of the training depends on the computational precision.

sized convolutions. This process allows us to define models VGG-22, VGG-25, VGG-28, etc. The number of parameters and training memory complexity of these models is shown in Table 4.1. Results in this table were computed on the assumption of a batch size of 1, input size of 32×32, and the output block (last 3 fully-connected/dense layers) consisting of 512, 256 and 10 units respectively. The remainder of the model matches the description in the original paper [114], with blocks of 64, 128, 256, and 512 convolutional filters with a spatial size of 3×3 and the relevant max-pooling between blocks. For simplicity, we assume the convolutional filters are zero-padded so the size of the input does not diminish.

The key point to note from Table 4.1 is that as the model gets bigger, the amount of memory required, both for parameters and for data storage, of end to end training increases linearly. With our proposed cascade learning approach, this is not the case; the total memory requirements is purely a function of the most complex cascaded sub-network (network trained in one iteration of the cascade learning). In the case of all the above VGG-style networks, this happens very early on in the cascading process. More specifically this happens when cascading the second layer, as can be seen in Table 4.2. The Table illustrates that after the second layer (or more concretely after the first max-pooling) the storage requirements of subsequent iterations of cascading reduces. The assumption in computing the numbers in this table is that the output blocks mirrored those of the end to end training and had 512, 256 and 10 units respectively.

If we consider Tables 4.1 and 4.2 together, we can see with the architectures in question that for smaller networks the end to end training will use less memory (although it is slower), whilst for deeper networks the cascading algorithm will require less peak memory and providing training time reductions. Given that the bulk of the space complexity for cascading comes as a result of the potentially massive number of trainable parameters in connecting the feature maps from the early convolutional layers to the first layer of output block, an obvious question is could we change the output block specification to reduce the space complexity for these layers? Experiments described in Section 4.2.1.1 explore the effect of reduced complexity output blocks on overall network classification performance.

4.2 Experiments

Our first experiment is performed on a less complex backpropagation problem and not on a CNN as explained in Section 4.1. We decided to execute this experiment to quickly determine the efficiency of Deep Cascade Learning. In this case we have chosen a small three hidden layer Multi-Layer Perceptron (MLP) applied to the flattened MNIST dataset. The results show that this algorithm is feasible and can obtain competitive generalization in early stages of the network with small improvements. This was a preliminary experiment, details can be found in the github repository.

To demonstrate the effectiveness of the Deep Cascade Learning algorithm, we apply it to two widely known architectures: a 'VGG-style' network [114], and the 'All CNN' [116]. We have chosen these architectures for several reasons. Firstly, they are still extensively used in the computer vision community, and secondly, they inspired state-of-the-art architectures, such as ResNets [45] and FractalNets [70]. Explicitly, the VGG net shows how small filters (3×3) can capture patterns at different scales. This is enabled by downsampling the images multiple times after sets of convolutions. The All CNN gave the idea of performing the subsampling with an extra convolutional layer rather than a pooling layer, and performs the classification using global average pooling and a dense layer to diminish the complexity of the network. The representations learned in each layer through end to end training are compared to the ones generated by Deep Cascade Learning. We compare our algorithm with an end to end model. The end to end model is trained using the standard procedure, and then we use the already learnt filters to train classifiers at every stage of the network (note that these filters are fixed given that they were learnt using end to end training). The training parameters of the models remain as similar as possible to make a fair comparison.

The learning rates in both experiments are diminished when the validation error plateaus. We evaluate the validation error after each epoch to determine whether the learning rate should be changed. More specifically, we use a mean-window approach that computes

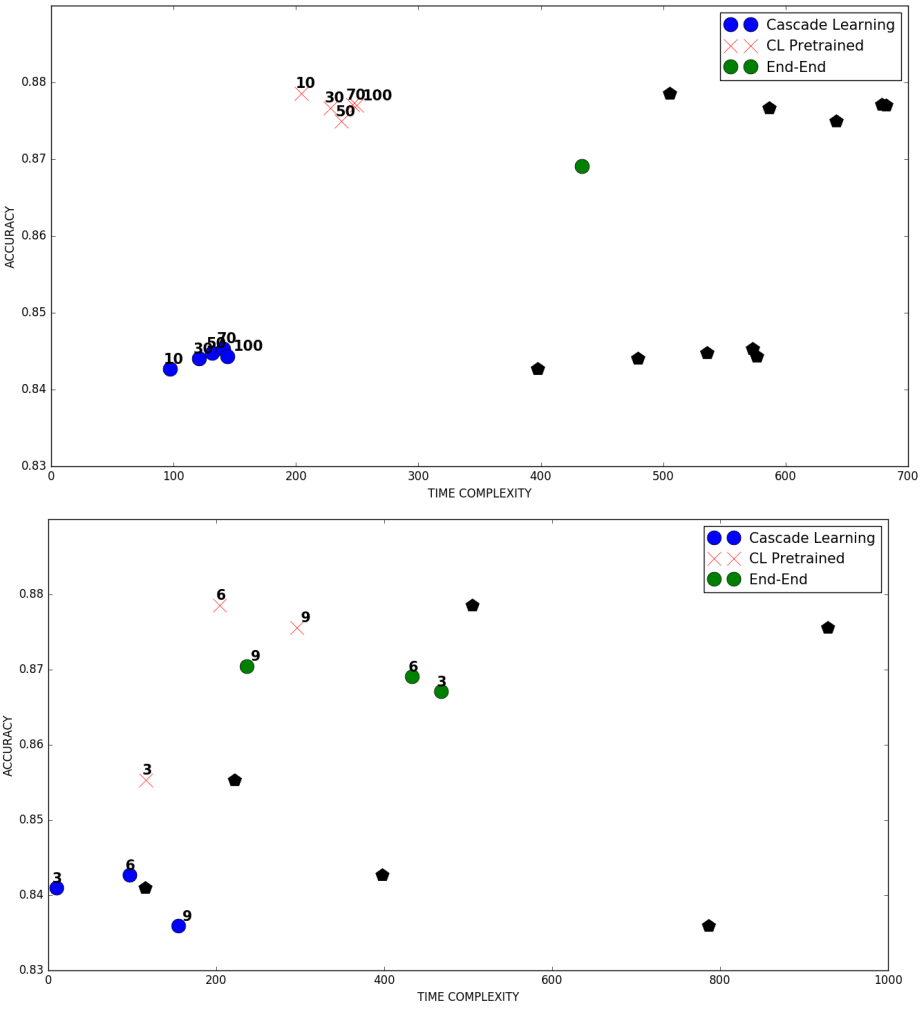

FIGURE 4.2: Time and depth comparison between Deep Cascade Learning, end to end, and pretrained using Deep Cascade Learning (see section 4.2.2.2 for details and results on pretraining using cascade learning). Multiple VGG networks were executed within a range of, (left) starting number of epochs (10-100) , (right) depth (3,6,9). Black pentagons represent runs executing the naive approach for both Cascade Learning and the pretraining stage. Solid blue dots represent optimal run, which caches the pseudo-inputs after every iteration.

the average of the last five epochs and the last ten epochs, and if the difference is negative then the learning rate is decreased by a factor of ten. The size of the window was tuned for the cascade learning only; if this approach is used in other training procedures it might be necessary to increase the size of the window.

The increase in the number of epochs in the cascade algorithm varies depending on the dataset. We performed experiments with an initial number of epochs ranging from ten to one hundred without any real change in the overall result, hence ten epochs as starting point is the most convenient. In all the experiments presented here, every layer iteration initialises a new output block, which in this case consists of two dense layers with ReLU activation [90]. The number of neurons in the first layer will depend on the dimensionality of the input vector, it may vary between 64 to 512 units, the second layer

contains half as many units as in the first layer. The final layer uses softmax activation and 10 or 100 units depending on the dataset, which results into an output block with three learnable fully connected layers. We did not observe major improvements by connecting a more complex output block, we illustrate this in Table 4.3.

4.2.1 Datasets

We have performed experiments using the CIFAR-10 and CIFAR-100 [66] image classification datasets, which have 10 and 100 target labels respectively. Both datasets contain 60k RGB 32×32 images split in three sets: 45k images for training, 5k images for validation, and 10k images for testing. In our experiments the datasets were normalized and whitened [67], however we performed no further data augmentation, following the procedure in AlexNet [67] and the Stochastic Depth paper [52].

4.2.1.1 CIFAR-10

VGG-style networks

We performed empirical evaluations on the weight decay and learning rate for Deep Cascade Learning. We took into consideration the fact that sub-models are single convolutional layer, which tend to easily overfit the data. Hence, tuning these two parameters was crucial to ensure convergence. In addition, we applied stochastic gradient descent with a starting learning rate of 0.01, and weight decay of 0.001. These values might still be suboptimal and further tuning could slightly improve performance. Our VGG implementation model contains six convolutional layers, starting with 128 3×3 filters and duplicating them after a MaxPooling layer. The initial weights remained the same in the networks trained by the two approaches to make the convergence comparable.

Space complexity and output block specifications. In order to test the memory complexity of this network we must take into account the output block specification. Specifically, we must consider the first fully connected layer, which in most networks contains the largest number of trainable parameters, particularly when connected to an early convolutional layer (see Section 4.1.4). On the first iteration of cascade learning, the output is $128 \times 32 \times 32$, hence, the number of neurons (n) in the first fully connected layer must be small enough to avoid running out of memory but without jeopardising robustness in terms of predictive accuracy. We have performed an evaluation by cascading this architecture with output blocks with a range of different parameter complexities. Table 4.3 shows the number of parameters of every layer as well as the performance for output blocks with first fully connected layer sizes of $n = \{64, 128, 256, 512\}$. In terms of parameters, cascade learning for early iterations can require more space than the entire

Training Regime.	Iter.	First output block unit count							
		64		**128**		**256**		**512**	
		param.	acc.	param.	acc.	param.	acc.	param.	acc.
CL	1	8.4e6	0.63	1.7e7	0.64	3.4e7	0.66	6.7e7	0.66
	2	8.5e6	0.69	1.7e7	0.72	3.4e7	0.72	6.7e7	0.73
	3	2.5e6	0.77	4.4e6	0.78	8.6e6	0.79	1.7e7	0.80
	4	4.5e6	0.80	8.7e6	0.81	1.7e7	0.81	3.4e7	0.81
	5	4.8e6	0.82	9.0e6	0.83	1.7e7	0.83	3.4e7	0.83
	6	1.6e6	0.83	2.7e6	0.84	4.8e6	0.84	9.1e6	0.84
End to end		2.8e6	0.86	3.9e6	0.87	6.0e6	0.87	1.0e7	0.87

TABLE 4.3: Number of parameters comparison using different output block specifications. Shows the effect of using between 64 and 512 units in the first fully connected layer (which is most correlated with the complexity). (Left) number of parameters, (Right) validation accuracy. Bottom row shows the parameters complexity of the end to end model. The increase in memory complexity on early stages can be naively reduced by decreasing the number of units of the fully connected layers. Potentially, memory reduction techniques on the first fully connected layer are applicable at early stages of the network. Later layers are less complex.

end to end network unless the overall model is deep. The impact of this disadvantage can be overcome by choosing a smaller n, and as shown in Table 4.3, the decrease of accuracy need not be particularly high when compared to the reduction in parameters and saving of memory.

As shown in Table 4.3, reducing the number of units can efficiently diminish the parameters of the network. However, we argue that in cases where the input image is massive, more advanced algorithms to counter the exploding number of parameters are applicable, such as Tensorizing Neural Networks [92] and Hashed Nets [14]. Based on the findings of these papers, applying those types of transformations to the first fully connected layer should not affect the results.

Training time complexity and relationship with depth and starting number of epochs. Equation 4.2 is dependent on the starting number of epochs i_l and its proportionality with depth. In Figure 4.2 we explored the effect of the time complexity by these two variables. To reproduce the left figure, several networks were cascaded with $i_l = [10, 30, 50, 70, 100]$. The overall required time is not drastically affected by i_l. For this particular experiment if $i_l > 50$, each iteration is more likely to be stopped early due to overfitting. The right figure shows the results on a similar experiment with varying network depth ($d = [3, 6, 9]$). Cascading shallow networks outperforms end to end training in terms of time. The epochs update constant (k in Algorithm 2) should be minimized on deeper networks to avoid an excessive overall number of epochs. Both figures show the importance of caching the pseudo-inputs, the black pentagons (naive run) are shifted to the right in relation to solid blue dots (enhanced run).

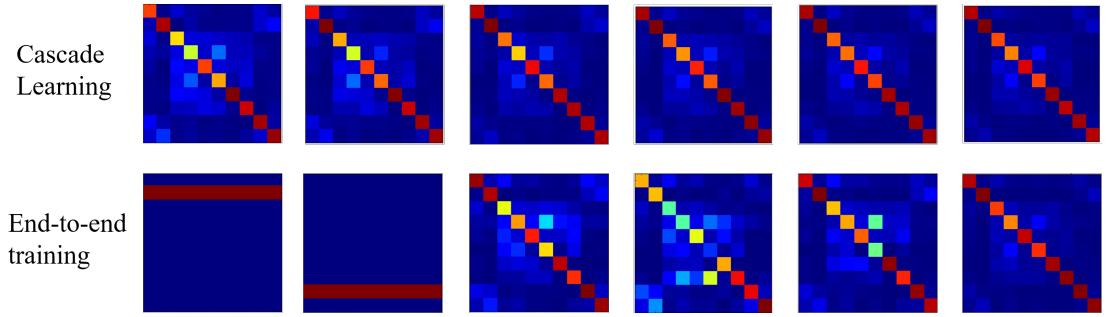

FIGURE 4.3: Comparison of confusion matrices in a VGG network trained using the cascade algorithm and the end to end training on CIFAR-10. First two layers of the end to end training do not show correlation with the output. While accuracy increases proportionally with the number of layer using the cascade learning, showing more stable features at every stage of the network.

Figure 4.3 shows confusion matrices from both algorithms across the classifiers trained on each layer. In this experiment we found that the features learnt using the cascade algorithm are less noisy, and more correlated with the output in most stages of the network. The results of the experiment show that the features learnt using the end to end training in the first and second layer are not correlated with the output; in this case the trained output block classifier always makes the same prediction. The third layer starts building the robustness of the features with an accuracy of 67.2%, and the peak is reached in the last layer with 85.6%. In contrast, with the cascade learning, discriminative features are learnt on every layer of the network. At the third layer, classes such as air plane and ship are strongly correlated with the features generated in both cases. The end to end training mostly fails to generalize correctly in classes related to animals.

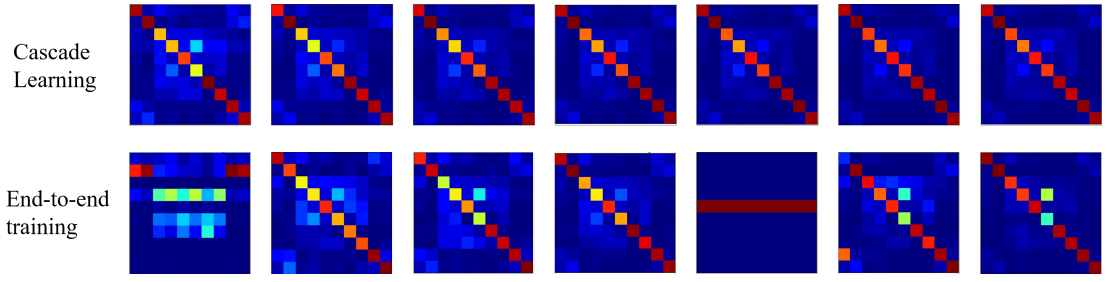

FIGURE 4.4: Comparison of confusion matrices in a The All CNN network trained using the cascade algorithm and the end to end training on CIFAR-10. Features learnt by cascading the layers are less noisy, and more stable.

On every iteration of the cascade algorithm, the sub-networks have a tendency to overfit the data. However, this is not entirely a problem since we only keep the convolutional layer for the overall model, which can be seen as pruning the model and ultimately

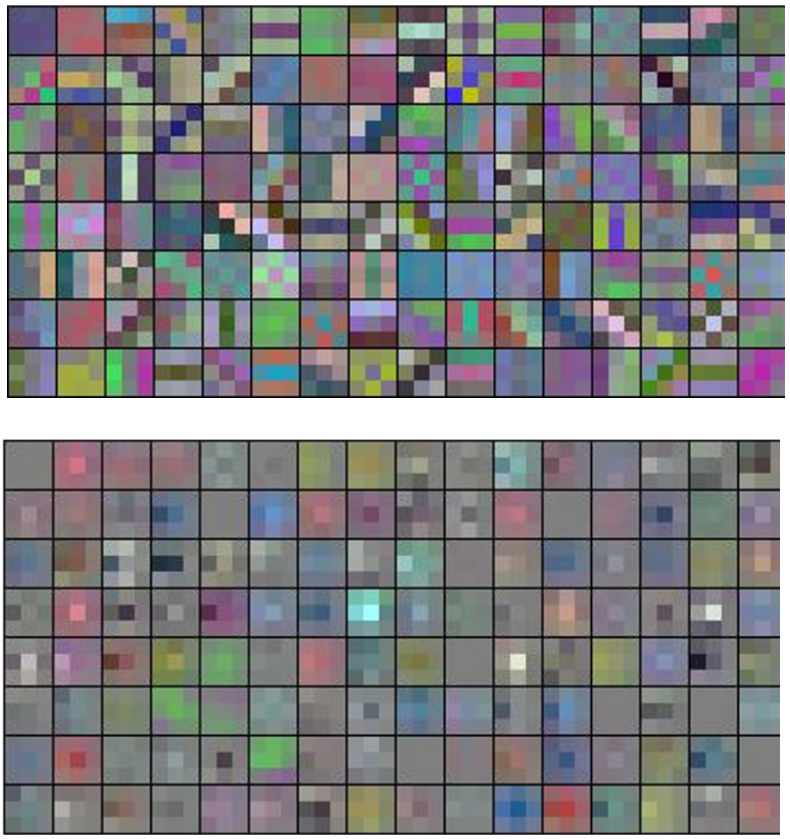

FIGURE 4.5: Visualization of the filters learnt in the first layer in both algorithms, (top) cascade learning, (bottom) end to end training. Each patch corresponds to one 3×3 filter. Filters learnt using the Deep Cascade Learning show different, and more clear, representations .

reducing overfitting of the overall model. This also avoids generating overfitted pseudo-inputs for the next iteration, hence disconnecting the dense layers works as a matter of regularisation.

One of the ways of determining if the vanishing gradient problem has been somehow diminished is by observing the filters/weights on the first layer (the one most affected by this issue). If the magnitude of the gradient in the first layer is small, then the filters do not change much from their initialized values. Figure 4.5 shows the filters learnt using both algorithms. The cascade algorithm learnt a range of different filters with different orientation and frequency responses, while using an end to end training the filters learnt are more redundant. Some filters in the end to end training are overlapping, this generates a problem since the information that is being captured is redundant.

It is naive to assume the problem is alleviated because the filters on the cascade learning are further apart from the initial filters. Hence, to complement the visualization of the filters, we calculated the magnitude of the gradient after every mini-batch forward pass on both cascade learning and end to end and plotted the results on Figure 4.6. For the end to end training, the gradient was computed at every convolutional layer for all the epochs. For the Deep Cascade Learning, the gradients were calculated on every iteration

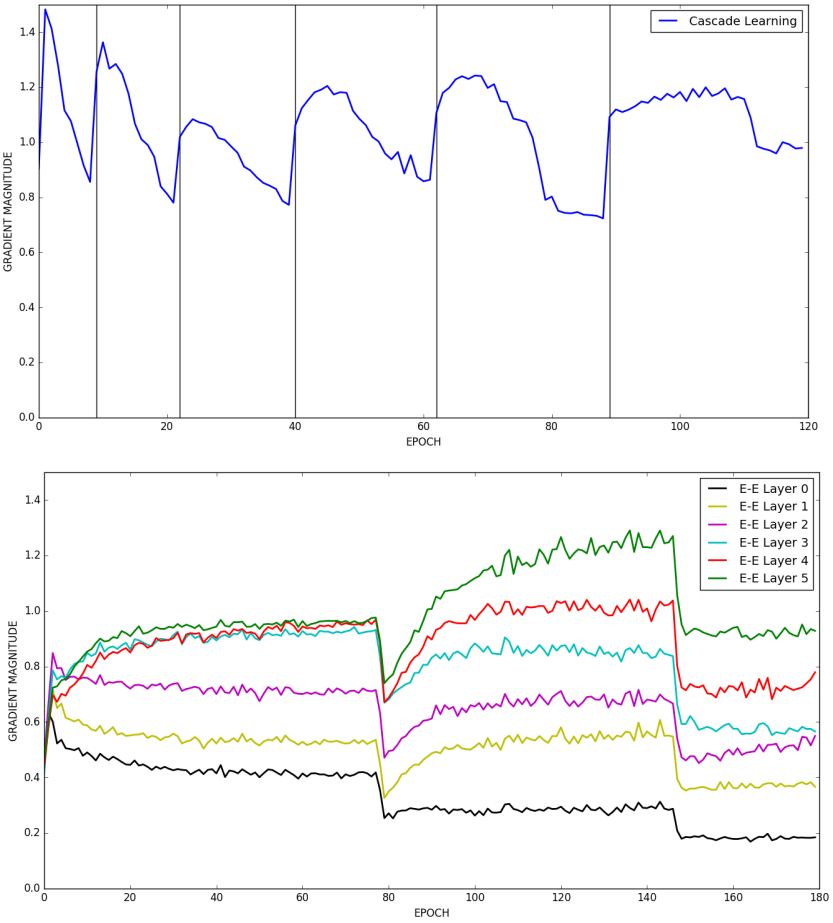

FIGURE 4.6: Magnitude of the gradient after every mini-batch forward pass on the convolutional layers of the end to end training (bottom), and the concatenated gradients of every Deep Cascade Learning (top) iteration. Vertical lines represent the start of a new iteration. Curves were smoothed by averaging the gradients (of every batch) on every epoch.

on the core correspondent convolutional layer. The curves are generated by averaging the mini-batch gradients in each epoch.

In contrast with Deep Cascade Learning, the magnitudes of the gradients in end to end training, on early layers, are substantially smaller than those on deeper layers. Overall, the gradients are higher for the Deep Cascade Learning. In addition, it requires fewer epochs with high updates on the weights to quickly fit the data on every iteration. With end to end training the opposite occurs; it requires more epochs (because of the small updates) to fit the data.

4.2.2 The All CNN

This architecture contains only convolutional layers, the downsampling is performed by using a convolutional layer with stride of 2 rather than a pooling operation. It

also performs the classification by downsampling the image until the dimensionality of the output matches the targets. The All CNN paper [116] describes three model architectures. We have performed our experiments using model C (deepest model in [116]) which contains seven core convolutional layers, and four 1×1 convolutional layers to perform the classification with an average pooling and softmax layers as the output block. In the case where the output block contains an average pooling and a softmax activation, each layer would learn the filters required to classify the data and not to generate discriminative features. Hence, to make a fair comparison of the filters we have changed the output block of the All CNN to three dense layers with softmax activation at the end. In the All CNN report [116] it is stated that changing the output block may result in a decrease in performance, however in this study we aim to widely compare both algorithms at every stage, rather than just final classification result. The number of parameters used when cascading this architecture varies between $2.7 * 10^6$ and $0.33 * 10^6$; on the other hand the end to end training requires us to store $1.3 * 10^6$ parameters.

The All CNN, using an end to end training, learns better representations on early layers than the VGG style net. The first convolutional layer achieves a performance of approximately 20% by learning three classes at most. This can be observed in the confusion matrix in Figure 4.4. In contrast with end to end training, the accuracy when cascading this architecture progressively increases with iterations, learning discriminative representations at every stage of the network going from 65% to 83.4%. It is important to note that, during end to end training, layer five did not learn representative features. However, this mysterious behaviour did not disturbed the accuracy on deeper layers.

Figure 4.7 compares the performance of both algorithms on each layer. The accuracy in the cascade learning increases with the number of layers. In addition, the variance of the performance is very low in comparison with the end to end. This is because it forces the network to learn similar filters in every run, decreasing the impact of a poor initialisation.

We have found that for a given iteration more than 50 epochs are not necessary to achieve a reasonable level of accuracy without overfitting the data. Additionally, we also tested the time complexity of this model within a range of starting epochs (similar experiment in previous section). These experiments went from 10 starting epochs to 50 (epochs increase by ten on every iteration with a ceiling on 50). The elapsed time for the All CNN model C is reduced by ~ 2.5 regardless of the starting number of epochs.

4.2.2.1 CIFAR-100

Similarly to the previous experiments, we have tested how the cascade algorithm behaves with a one-hundred class problem using the CIFAR-100 data set [66]. The experimental

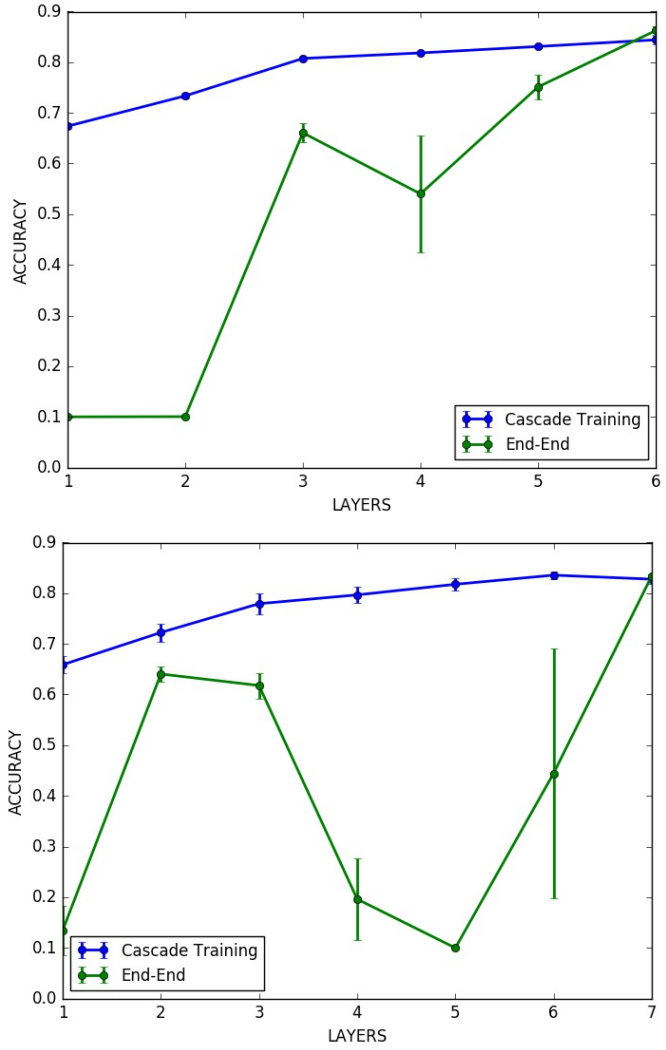

FIGURE 4.7: Performance on every layer on both architectures, (top) VGG, (bottom) The All CNN. Cascade learning has a lower variance making the initialization less relevant to the classification at each layer. It also shows a progressive increase in the performance without the fluctuations presented in the end to end training.

settings remain the same as the previous section, and the main change to the model is that the output layer now has 100 units to match the number of classes.

In a VGG-style network, the comparison between both algorithms is similar to a ten class problem. In end to end training, the first two layers do not learn meaningful representations, and each layer learns better features using the cascade algorithm. However, the end to end training performs better by approximately 1% on the final classification.

In The All CNN Network, the features learnt in the end to end model remained more stable than in CIFAR-10. Similarly to the previous experiment, the first four layers were outperformed by the cascaded model. However, the end to end model had better performance over by 3% and 6% on the last layers.

		1	**2**	**3**	**4**	**5**	**6**	**7**	**Fine-Tune**
VGG	CL	0.35	0.39	0.50	0.50	0.53	0.59	-	**0.63**
	E-E	0.01	0.03	0.22	0.14	0.35	0.60	-	
The All CNN	CL	0.31	0.39	0.47	0.46	0.49	0.54	0.52	**0.67**
	E-E	0.03	0.05	0.03	0.41	0.54	0.61	0.62	

TABLE 4.4: Comparison of accuracy per layer using the cascade algorithm and end to end training on CIFAR-100 in both architectures. Using the cascade learning outperforms almost all the layers in a VGG network, and almost achieves the same accuracy in the final stage. The All CNN with an end to end training outperforms in the final classification, however the first three layers do not learn strong correlations like when using the cascade learning.

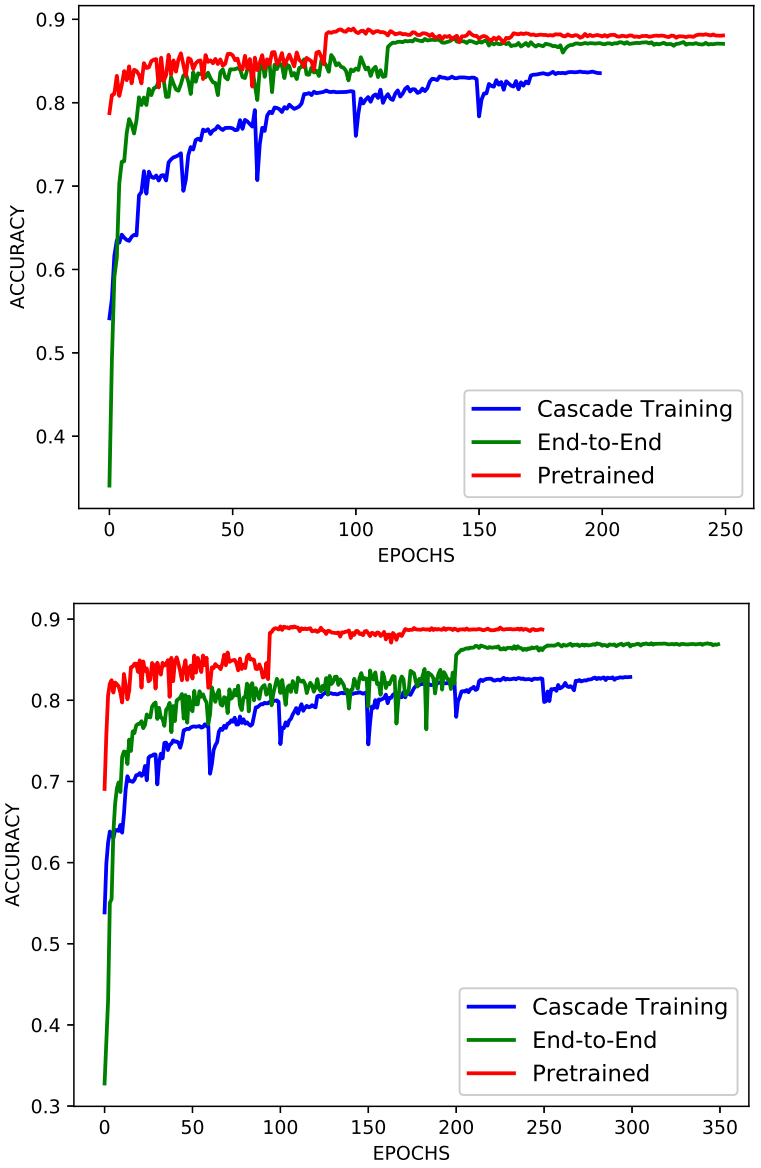

FIGURE 4.8: Performance comparison on CIFAR-10 between pre-trained network and random initialization, (left) VGG, (right) The All CNN. The step bumps in the Cascade learning are generated due to the start of a new iteration or changes in the learning rate.

The results on a 100 class problem are arguably the same as in a ten class one. It is noted that The All CNN Network, when trained end to end, can outperform the cascade algorithm in the final classification but not in the early layers. In the VGG style network, Deep Cascade Learning build more robust features in all the layers, except for the last layer which had a difference of 1%. Table 4.4 shows a summary of the results on every layer for both algorithms. Early robust representations are useful to perform early and "quick" inference. In addition, these early layers may contain more information (given the better performance) that can potentially be used to transfer features. However, it does not improve accuracy in deeper stages. We can argue that this comes as a result of early overfiting of Deep Cascade Learning due to domain specificity at early layers. I

4.2.2.2 Pre-training with cascade learning

In the experimental work described so far, the main advantages of cascade learning come from: (a) reduced computation, albeit at the loss of test accuracy in comparison to end to end training; and (b) a better representation at intermediate layers (multiple early classifiers, and potentially transfer learning). We next sought to explore if the representations learned by the computationally efficient cascading approach could form good initialisation of end to end trained networks and achieve performance improvements.

The weights are initialised randomly. Then the procedure is divided into two stages: firstly, we apply DCL to the from the bottom to the top of the network. We then sequentially connect all the layers to obtain a full model. In the subsequent stage, the network is fine-tuned using a backpropagation and stochastic gradient descent, similarly to the end to end training. We applied this technique to both VGG style network and The All CNN. For more details on the architectures refer to Section 4.2.1.1.

Figure 4.8 shows the difference in performance given random and cascade learning initialisation. The learning curves in the figure are for the VGG and The All CNN architectures trained on CIFAR-10. The improvements in testing accuracy varies between $\sim$2 to $\sim$ 3% for the experiments developed in this section. However, the most interesting property comes as a consequence of the variation of the resulting weights after executing the Deep Cascade Learning. As shown in the previous section, this variation is significantly smaller in contrast with its end to end counterpart. Hence, the results obtained after pre-initializing the network are more stable and less affected by poor initialization. Results on Figure 4.2 show that even including the time of the tuning training stage, the time complexity can be reduced if the correct parameters for the cascade learning are chosen. It is important to mention that end to end training typically requires up to 250 epochs, while the tuning stage may only require a small fraction since the training is stopped when the training accuracy reaches ~ 0.999.

The filters generated by Deep Cascade Learning are arguably overfitted (first layer typically achieves $\sim 60\%$ on unseen data and $\sim 95\%$ on training data) as opposed to the end to end training, on which the filters are more likely to be close to their initialisation. By pre-training with cascade learning, the network learns filters that are in between both scenarios (under and overfitness), this behaviour can be spotted on Figure 4.9.

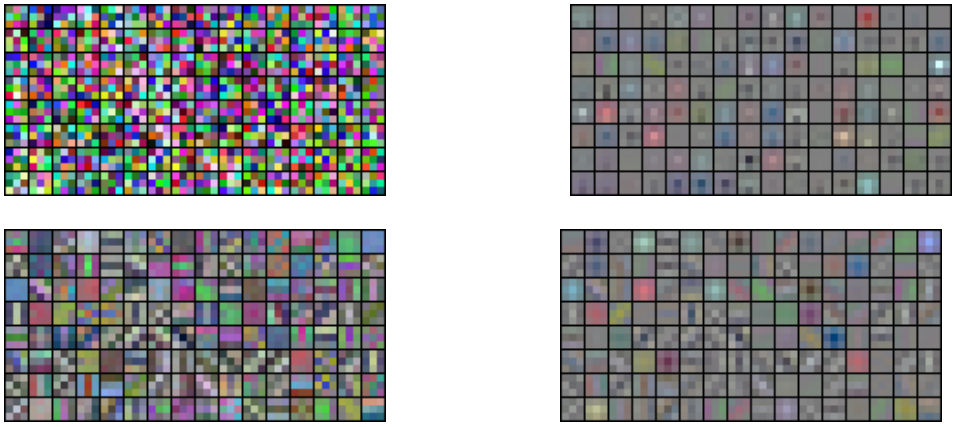

FIGURE 4.9: Filters on first layer for at different stages of the procedure on the VGG network defined in the previous section. (top-left) initial random weights, (top-right) end to end, (bottom-left) cascaded, (bottom-right) end to end trained network initialized by cascade learning. The images were normalised and resized accordingly.

Figure 4.10 shows the test accuracy during training of a cascaded pre-trained VGG model on CIFAR-100.

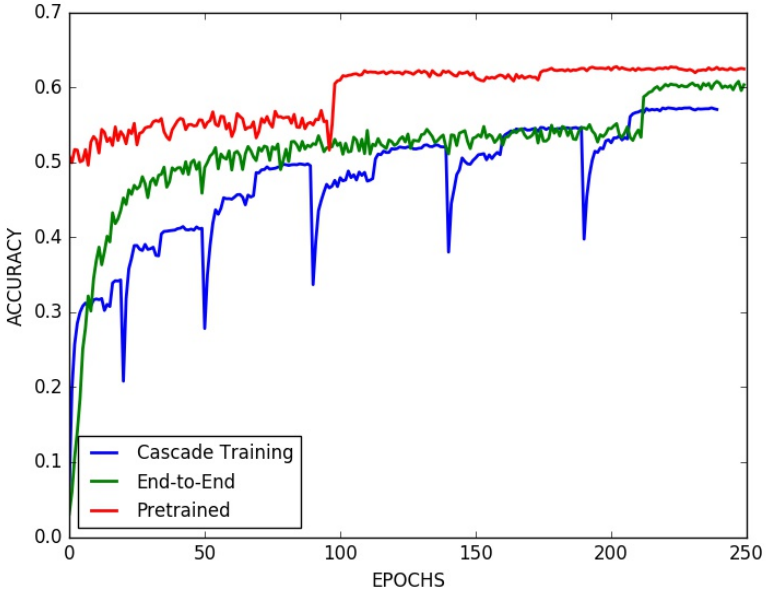

FIGURE 4.10: Performance comparison between pretrained network and random initialization on CIFAR-100 using a VGG network.

4.3 Summary

In this Chapter we have proposed a new supervised learning algorithm to train deep neural networks. We validate our technique by studying an image classification problem on two widely used network architectures. The vanishing gradient problem is diminished by our Deep Cascade Learning. This is as a result of decreasing intermediate layers of every submodel, which ultimately allocates output blocks adjacent to the layer to train. it is focused on learning more robust features and filters in early layers of the network. In addition, the time complexity is reduced because it no longer needs to forward propagate the data through the already trained layers on every epoch. In our experiments the memory complexity is decreased by more than three times for the VGG style network and four times for The All CNN. Standard end to end training has a high variance in the performance, meaning that the initialization plays an important role in ensuring a good minimum is reached by each layer. Deep Cascade Learning generates a more stable output on every stage by learning similar representations at every run. In addition, the Deep Cascade Learning algorithm has been demonstrated to scale to 10 and 100 class problems, and shows improvements in the features that are learnt across the stages of the network. Using this algorithm allows us to train deeper networks without the need to store the entire network in memory.

As a summary of the findings of Deep Cascade Learning, we can conclude:

- Memory and training time reductions. Saving the pseudo-inputs of already trained early layers can decrease memory and time during training. This avoid the need of forward propagating across already trained layers.
- Early classifiers contain important information about the data and can be used to make early predictions. Potentially, an ensemble model can be created by stacking predictions at of all this multi-scaled classifiers.
- Given that early layers can also be used to perform a classification, early layers could be included in the transfer learning field. This can potentially lead to more transferable networks, where early layers might hold more information about the target domain. This would not be the case for the end to end training.
- It has to be mentioned that this algorithm has been tested on very closely related datasets. This gives us an intuition of its applicability, but it does not mean that it is scalable to every machine learning field. We encourage the community to make use of Deep Cascade Learning (or supervised greedy layer wise algorithms) on multiple machine learning tasks. This could lead to increased scalability of Deep Cascade Learning applications.

It should be noted that our algorithm is not aimed at obtaining better performance than standard approaches, but with significant reduction in the memory and time requirements. We have shown that if improvements in generalization are expected, this algorithm can be used as a pre-training algorithm technique.

There are many questions that are still yet to be answered. How deep can this algorithm go without losing robustness? We believe that if the performance cannot be improved by appending a new convolutional layer, l, it should at least be as good as in the previous layer, $l-1$, by learning filters that directly map the input to the output (filters with 1 in the centre, and zero in the borders). This might not happen because the layer might quickly find a local minimum. This could be avoided with a different type of initialization; most probably one specialised for this algorithm. In Deep ResNets [45], the accuracy diminished when they went beyond 1200 layers, and hence the vanishing gradient problem was not entirely circumvented. We believe this algorithm might be able to go deeper without losing performance by partially overcoming the vanishing gradient problem, learning "mapping" filters to maintain the features sparseness, and learn a bigger set of high level features. In addition, Deep Cascade Learning has the potential to find the number of layers required to fit a certain problem (adaptive architecture), similarly to the Cascade Correlation [28], Infinite Restricted Boltzmann Machine [21], and AdaNet [20].

Chapter 5

Cascade Transfer Learning

In Chapter 2 we discussed transfer learning as an approach to applying deep learning when data availability is limited. In addition, Chapter 3 explores layer-wise training methods for non transfer learning problems. In this chapter we discuss a novel approach to transfer features from one dataset to another. This transfer learning technique shares several similarities with the Deep Cascade Learning (see Chapter 4). Typically, transfer learning is applied by using the feature extractor at the last stage of the pre-trained network. This is assuming that the latest feature in the network holds the most information. We propose a framework to gradually learn transferable features across the network. In addition, this approach also provides an insight to where in the network features are most transferable to the target dataset. This can lead to a more efficient and less complex network in comparison with traditional deep transfer learning.

The code used to generate the results of this Chapter can be found in http://github.com/EnriqueSMarquez/CascadeTransferLearning

5.1 Background

Transfer learning has been explored to share features extractors across similar, yet different domains [22, 85]. The methodology commonly uses two datasets, a *base* dataset, typically a balanced dataset (in order to obtain less biased machine learning algorithms) with a large amount of data; and a *target* dataset, which may contain less data and different labels. It is motivated by the robustness of the feature extractor, and the idea that filters are reusable on objects with shared properties. Kim et al. [62] proposes a framework to transfer features when the base and target datasets are not in a similar space. Their framework uses a *bridge* dataset to adapt the feature extractor to a closer space to the target dataset. This avoids losing generalization due to domain specificity.

This is applicable when the base dataset is very different from the target dataset. Applications of transfer learning include, medical imaging [62], soft-biometric [83], semantic segmentation [103], and were further discussed on Chapter 2.

Yosinski et al. [132] measures feature transferability between subsets of ImageNet [109]. As opposed to the typical approach where features are transferred at the last stage of the network, they chopped the network at every stage and trained a dense layer to quantify the robustness of the features as the network gets deeper. Therefore, measuring the transferability of the features at every stage of the network. They concluded that fine-tuning recovers co-adapted interactions between layers. Fine-tuning is a crucial step to better transfer features regardless of the layer of the network on which it was chopped.

Kornblith et al. [65] tested 13 state-of-the-art models on 12 datasets. In their experimental setup they re-trained the last layer (classifier), and fine-tuned the feature extractor. They concluded that better feature extractors for ImageNet yield better results on target datasets. This result is intuitive since the 12 datasets used have similarities to ImageNet's labels, hence, the generalization on the target dataset is proportional to ImageNet.

5.2 Cascading pre-trained networks

In this section we propose a new algorithm that we call Cascade Transfer Learning (CTL). We explain the algorithm, its complexity, and properties.

The CTL algorithm starts from the pre-trained network just like traditional transfer learning. It then sequentially trains hidden layers of the network. Every iteration requires an initialised classifier, hence, CTL outputs as many classifiers as layers trained (iterations). This helps early layers to adapt to the target domain instead of fully preserving the representations learnt through the base dataset. In addition, CTL aims to determine at which stage the network comprises the most information about the target domain. In our CTL executions, we start the algorithm at mid-stages of the network to avoid losing key coarse representations of the base dataset. In addition, the algorithm reuses early layers features (already trained features) by appending skip connections from these stages to the layer to train. The rational behind these decisions are explained in later sections when we explore the hyperparameters of this algorithm.

In Figure 5.1 we show the steps required to perform CTL. The diagram makes use of a five layer ResNet (as illustration) and shows the flow of CTL. The algorithm starts at the third residual block, and propagating one extra residual connection. On the first stage, the model is trained end to end on the base dataset, in order to obtain the feature extractor to transfer. Afterwards, CTL starts by training a module of the pre-trained model, the algorithm can potentially start at any stage of the network. We evaluate the

influence of the starting layer in later sections. On each iteration, the algorithm trains a pre-trained residual block and discards/stores classifiers from previous iterations. These extra residuals allow the network to progressively adapt each layer to the target domain while preserving the synergy between the layers. This can be achieved by avoiding overfitting early layers. A discussion on how to tackle overfitting of cascaded models can be found in Chapter 4.

The algorithm has several similarities with Deep Cascade Learning. Firstly, it adapts the pre-trained network layer by layer and freezes already trained layers. The cost of propagating the inputs through already trained layers is equivalent to one forward pass through the cascaded layers. If certain accuracy needs to be achieved while minimizing resources, the algorithm can be stopped at any given stage of the network. CTL also uses the same framework as Deep Cascade Learning, with the difference that the network is initialised using pre-trained weights instead of random initialisation. Additionally, techniques that are applicable within the CTL algorithm are also applicable to Deep Cascade Learning. For example, the dimensionality reduction block to decrease the number of parameters on each iteration could be applied in Deep Cascade Learning.

$$\hat{H}_i = ReLU(\mathbf{w}_i^t H_i + b_i) \tag{5.1}$$

$$y = Softmax\Big(\mathbf{w}_{cl}^t \sum_i \hat{H}_i + b_{cl}\Big) \tag{5.2}$$

We explored multiple approaches to combining the residual connections. Our first approach would concatenate the feature vectors from the residual connections and trains the classifier on this complex feature, however, this leads to a linear increase in the number of parameters of the classifier. Figure 5.2 shows the chosen connectivity diagram for one iteration of CTL, where H_n represents the features at stage n. To overcome the dimensionality issue, we pool the features and connect a linear layer with an activation function to reduce the dimensionality and match H_n. We then add these features to obtain a single feature vector per input, which is later classified using an MLP with softmax activation. If the network already contains residual connections, we start propagating from H_{n-2} to avoid redundancy in the features. Equations 5.1 and 5.2 illustrate the logic to generate H_i, this dimensionality reduction/matching layer can be multiple dense layers, or a single dense layer. Each branch contains its own classifier with parameters w_i (linear weights), b_i (bias) and its evaluated using a ReLU activation. The prediction y outputs a probability vector, and w_{cl}, b_{cl} the parameters of the classifier. In our experiments we have found that the dimensionality of H_i can be further reduced by connecting a 1×1 convolutional layer before pooling. This does not affect the performance of CTL.

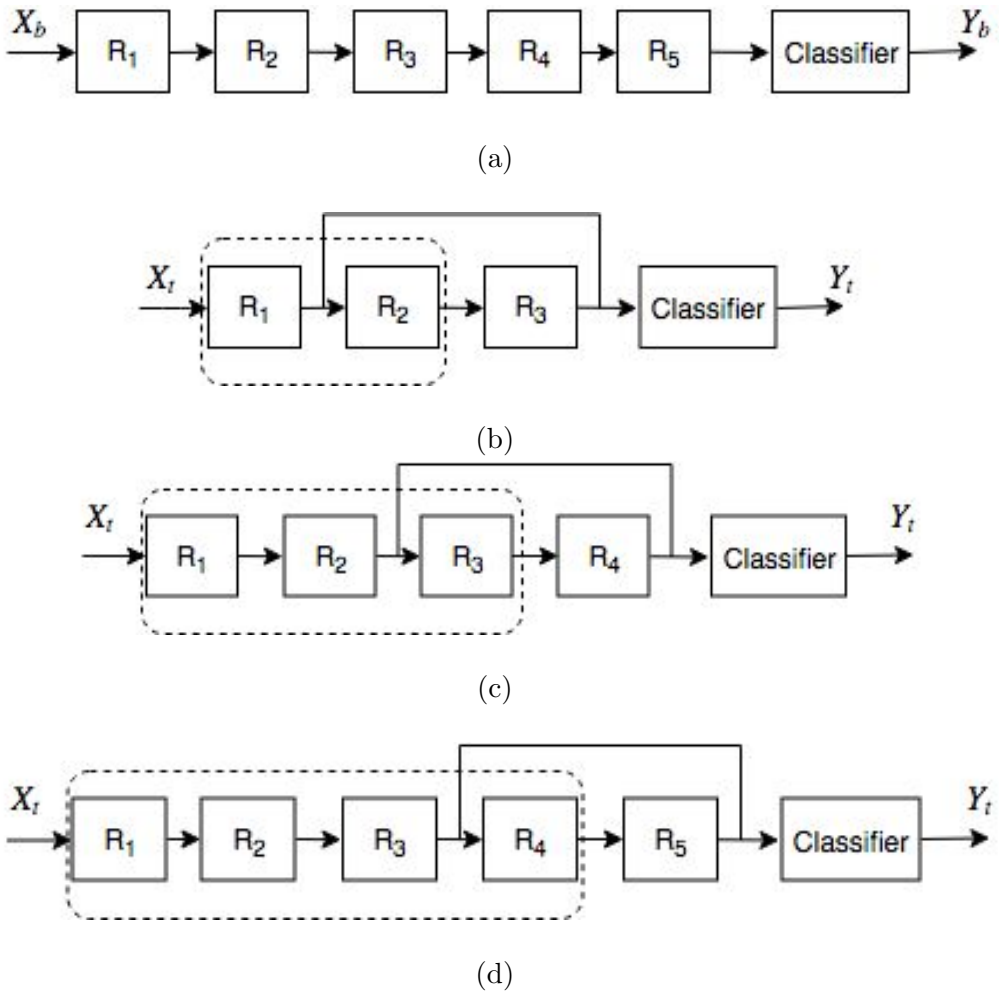

FIGURE 5.1: Diagram illustrating the steps of CTL. For simplicity, the diagram uses a ResNet with five residual blocks, the algorithm starts at layer two, and on each iterations in connects an extra residual. On the first stage of CTL (a), the chosen architecture is trained end to end on the base dataset, thus, resulting in a model with learnt weights on the base dataset. Then it iteratively cascades modules of the pre-trained model while freezing previously cascaded layers (b), (c), and (d). Dashed boxes represent frozen modules. On each iteration the classifier can be stored for later predictions or discarded.

5.2.1 Algorithm Complexity

On each iteration the model trains a single classifier and residual block. The largest fraction of the parameters reside in the classifier and fully connected layers. Their complexity depends on the dimensionality of the features.

Following [82], the algorithm can be implemented in two ways depending on the hardware, implementation, and framework flexibility. Naively, one can have the whole network loaded on the device (typically GPU) and fix the required layers. Thus, there is an additional cost of having the need to propagate through the network, however, the gradients do not have to be stored. This implementation is more flexible and allows

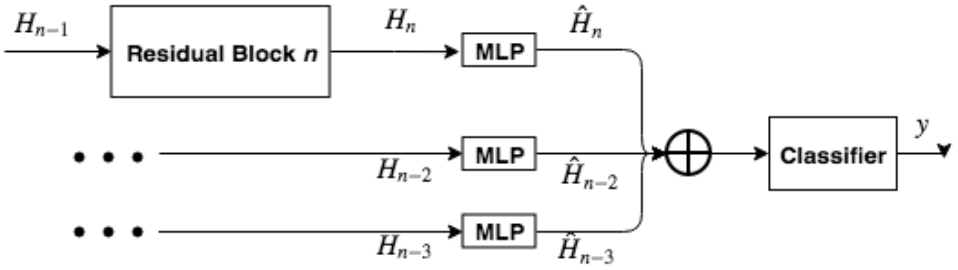

FIGURE 5.2: Model for iteration n of CTL including two residual connections. H_n represents the hidden state at stage n. Previous hidden states are connected to learnable MLPs to match the dimensionality H_n. The resulting vectors are then added together to hold a single feature vector. The last stage takes this vector and classifies it using a linear layer with softmax activation. The features can be spatially pooled before each MLP to further reduce the complexity of the algorithm. The layers or blocks are already pre-trained on the base dataset.

us to test different executions. The optimal algorithm caches the feature vectors at the required stages and only loads the model and classifier that are trained on that iteration, hence, the cost is reduced to only that of propagating through a residual block and its classifier.

Depending on the implementation, the memory complexity of CTL can be minimized to a fraction of its fine-tuned counterpart. Additionally, every classifier is trained to generalize on the target domain. This generalization is increased as the network adapts more layers. We can use early classifiers to do inference on 'easy" (or far apart from the boundaries) inputs and discard the need of propagating until the last layer. This can be applied to minimize the hardware cost if false positives are not crucial.

The time complexity depends on the number of epochs required to fit the dataset for both fine-tune network and CTL. Thus, we explore this empirically on later sections.

5.3 Algorithm Hyperparameters

Besides the training and network variables (e.g. learning rate, number of filters, momentum, weight decay), CTL contains two hyperparameters:

- The starting stage. The algorithm may start at any stage of the network, from layer 0 to layer n. However, as mentioned in the previous section, in our experiments we concluded that cascading early layers can disturb the hierarchical structure between layers. This ultimately affects the convergence of the network by forgetting important early representations. Yosinski et al. [132] further discusses this behaviour and concludes that this issue can be alleviated by fine-tuning the model.

However, our algorithm aims to avoid the need of fine-tuning the model. Taking this into account, we start the algorithm half-way through the network.

- Residuals from previous layers. Potentially one could connect a residual connection from every previous layer. This can increase the number of parameters drastically as well as adding redundancy to the resulting feature vector.

We present a study on the influence of these hyperparameter in later sections.

5.4 Using early classifiers for resource efficiency

Resource efficient recognition has been widely addressed by the computer vision community. Jones and Viola [59] proposed an algorithm to efficiently discard false positives using weak early classifiers. The idea was further explored using deep networks in the Multi-Scale resource efficient network [50]. This network can classify 'easy' images at early stages, and more complex images on later layers. The classifier to use directly depends on a fixed uncertainty threshold, which is fixed by the programmer depending on the performance of each classifiers.

Similarly, Cascade Transfer Learning learns a classifier per stage. We can stop the inference if a prediction holds a greater probability than the fixed threshold. This leads to a resource efficient network where inputs may not propagate through the entire network, but exit at early stages.

5.5 Experimental Setup

In this section we discuss the datasets, models, and methodology applied on this chapter. It is worth mentioning that the chosen datasets do not have enough images to train deep networks from scratch, as doing so quickly overfits the data.

5.5.1 Datasets

Kornblith et al. [65] presents results of applying transfer learning on several datasets. For simplicity, we only chose three of these datasets. The datasets were randomly selected from [65] taking under consideration that one of the datasets had to be far apart from the base dataset. The chosen datasets are described as follows:

- **Caltech-102** [29]. Includes a total of 9144 images with 102 categories, such as faces, animals, food, and vehicles. The dataset contains 40 to 800 images per

class. There are no training or test set defined, hence in order to obtain similar datasets as the literature, we show results on this dataset by performing 3-fold cross-validation five times (defined as Repeated Stratified Crossvalidation). Splitting the data using this methodology yields training and test sets with 3060 and 6084 images respectively. Each fold of the evaluation method is balanced to decrease the variance of training. Considering the imbalance in classes, we test using mean-per-class accuracy as otherwise the results are misleading.

- **Flowers-102** [91]. Contains 102 classes of commonly encountered flowers in the United Kingdom. Each label holds from 40 to 258 images. Nilsback and Zisserman [91] includes one split for test, training, and validation set. Similarly to the Caltech-101 dataset, we tested using a mean-per-class accuracy metric to take into account the imbalance of the dataset.

- **Describable Texture Dataset** (DTD) [16]. Consists of 5640 images with 47 labels inspired by human perception. The dataset is perfectly balanced with 120 images per class. Cimpoi et al. [16] include ten splits for training and test set. In our experiments on this dataset, we train and test using all ten splits and average the performance to obtain a more general metric. This dataset is less similar to ImageNet which makes the transferred features less robust for the target dataset.

5.5.2 Models

Kornblith et al. [65] tested several pre-trained models and measured their feature transferability. We chose to test our algorithm on ResNets, given that their conclusions are that ResNets are the best feature extractors to apply transfer learning on [65]. They compare ResNets with several other architectures that are capable of obtaining a better ImageNet top-1 accuracy. However, features learnt with ResNets include better transferability given that they outperform the remainder of the models on most of the studied datasets [65]. Their conclusion was merely empirical and, even though was tested on several datasets, can still be a biased conclusion. We can make use of their conclusion given that we are using the same datasets.

We directly compare traditional transfer learning with CTL. Each iteration of CTL trains one residual block. More specifically, we selected the pre-trained networks as ResNet-34, ResNet-50, and ResNet-102 trained on ImageNet. This decision was made out of simplicity, since these three networks can be fine-tuned in a single graphics card (NVIDIA GTX 1080 used for experiments), whilst any deeper network requires at least two devices to fine-tune.

ResNet-34 uses a basic residual block structure with two convolutional layer with a kernel size of 3. In contrast, ResNets-50/101 require a bottleneck structure to reduce the dimensionality of the block. The bottleneck residual block includes three convolutional

layers, the first one with 1×1 convolutions and a decrease of the number of filter by 4. The second layer is a traditional 3×3 convolution similar to the ones used in the basic block. Finally, the third layer expands back the number of filters by 4 with a 1×1 convolution. This technique is applied to reduce the complexity of the network and enables the possibility of going deeper without drastically affect the number of parameters.

The data to train on the base dataset is normalized, and the parameters (mean and standard deviation) are stored. When transfer learning is applied, the data of the target dataset is normalized accordingly to the normalization parameters used to pre-train the network. If this normalization is not applied, the feature space of the target dataset will be shifted from the ImageNet feature space, which will induce bias on the features. Moreover, this normalization can be seen as reducing the covariance shift of the input data from the target dataset to the base dataset.

5.6 Measuring transferability

In this section we compare our Cascade Transfer Learning algorithm with two commonly used approaches for transfer learning: fine-tuning, by training the last classifier for several epochs followed by unfreezing the encoder to tune the filters to the target domain. By fine-tuning, ResNets have shown state-of-the-art results on transferring features. For each dataset we tune the hyper parameters using the train and validation set. We then execute runs with tuned hyperparameters on both training and validation set. The testing is performed after the network has been trained, therefore we do not include any early stopping. We believe that by sequentially decreasing the learning rate, the network converges to a minimum and the accuracy on unseen data does not decrease. Therefore, exiting this local minimum is not feasible for the model due the learning rate at this point being is too small.

Both fine-tuning and CTL algorithms are trained using Stochastic Gradient Descent (SGD) with momentum. Following the literature [65, 132], we constrain the cross-entropy loss function with weight decay of 10e-4. For the fine-tuning case we trained the classifier for 50 epochs, dropping the learning rate by a factor of ten at 30 and 45 epochs, and trained the entire network for an additional 30 epochs with a similar learning rate schedule at 15 and 25 epochs. Subsequently, we train each sub-network of the CTL for 30 epochs each, and decrease them by the same factor at 15 and 24 epochs. We did not observe improvements when applying data augmentation to any of the datasets, hence, all our results are shown without further augmenting the data. The batch size for all the experiments is 32. These training hyperparameters were either empirically tuned or taken from the literature of the pre-trained networks, more importantly, they remain the same on both experiments to make a fair comparison.

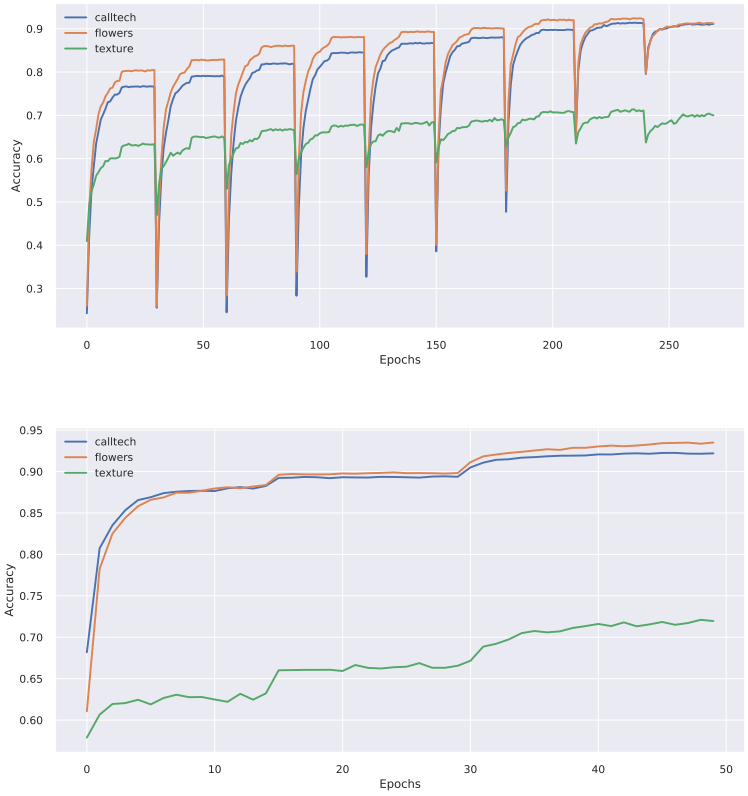

FIGURE 5.3: Accuracy comparison for ResNets-34/50/101 tested on the datasets of study. Top curves represent CTL, bottom curves traditional fine-tuning transfer learning.

In our experiments we found that using two residuals from previous layers (layers H_{n-3} and H_{n-2}) are enough to boost the performance of CTL. In addition, the learning rate is divided by ten for the residual block in comparison to the classifier. This is because we want the algorithm to quickly learn the classification while gradually learning a better representation for the target dataset on early stages of the network. We have found that if any given iteration of CTL is executed with high learning rate, the robustness of the transferred features may be distorted, which leads to a drastic decrease in performance.

Table 5.6 compares our transfer learning algorithm with traditional fine-tuning. These results are generated by applying cross-validation across the dataset. For the Caltech dataset, we applied a balanced cross-validation to avoid generating splits with imbalanced classes. We applied 3-fold cross-validation five times in order to preserve the number of samples ratio (training 25% and testing 75%) and generate a similar dataset size that matches the literature. This yields a training set of 2040 images and a test set of 6149 images. The Flowers-102 dataset provides specific validation and test splits. Hence, we tuned the network using the both training and validation set, to then retrain on both datasets and evaluate on the given test set. This procedure was performed several times and the results were averaged. For the remaining texture dataset, we

	Caltech		Flowers		Texture	
	CL	F-T	CL	F-T	CL	F-T
ResNet-34	0.90	0.89/0.92	0.89	0.88/0.93	0.67	0.63/0.7
ResNet-50	0.91	0.89/0.92	0.91	0.90/0.93	0.71	0.67/0.72
ResNet-101	0.92	0.90/0.93	0.91	0.90/0.93	0.71	0.66/0.72

TABLE 5.1: Performance comparison on test data of Cascade Transfer Learning and fine-tuning Transfer Learning. Fine-tuning involves two stages, left column shows performance of training the last classifier with the rest of the network frozen, right columns shows after the network weights are tuned. The results are generated by applying cross-validation

cross-validated using all ten given splits of the dataset. We then again, averaged the results to obtain a more general performance.

It can be seen that CTL achieves competitive (within the standard deviation of the models) results in comparison to fine-tuning the transferred network in an end to end fashion. More importantly, our algorithm benefits from deeper networks without additional memory requirements, this is further investigated in section 5.8. In addition, results on the Texture dataset suggests that CTL is more successful on those target datasets that are more different from the base dataset. The average performance variance is very small for both algorithms, we believe this is because there is not much randomness in the algorithm since the initial weights are always the same.

Figure 5.3 shows the performance of ResNet-50 trained using CTL and fine-tuning at every epoch. We can observe that early classifiers can converge to a competitive accuracy. The use of residual connections allows learning better representations at every stage. These plots suggest that a deeper feature does not necessarily mean a more transferable feature. Hence, residual connections allow to propagate the transferability of the features with each cascade iteration. Several experiments were performed without residual connections, however, they were not as successful due to losing coarse representations. In Figure 5.4 we can spot the decrease in performance when not appending residual connections between cascaded layers. Considering Figure 5.3, we can also observe that for all the datasets maximum performance might not be achieved at the last stage of the network. Thus, CTL can find at which stage the features are best transferred. For the specific datasets of study, the CTL trained models generalized better at stage 9, 8, 7 for Caltech-102, Flowers-102 and DTD respectively. The incremental performance across layers is not observed when alone tuning a classifier at every stage of the network. Yosinski et al. [132] discusses this behaviour, and they concluded that it is due to representation specificity and fragile co-adaptation of features.

5.7 Effect of the number of residuals & starting stage

In this section we evaluate how the hyperparameters of the algorithm affects its convergence. The training parameters remained the same as in section 5.6. We did not perform cross-validation due to the high time complexity of the experiment, which would require 15 crossvalidation runs over a parameters space of 6 on 3 datasets ($15 \times 6 \times 3 = 270 runs$). The following results are on validation data alone.

5.7.1 Number of residuals

For this experiment we fixed the starting stage at half of the network depth. We then proceeded to execute CTL on ResNet-50 with different number of residuals and quantified performance at every iteration of CTL. It is worth mentioning that incorporating a new residual requires a new MLP, which can increase the number of parameters.

The number of residuals is crucial to preserve the feature robustness across every layer. Therefore, adding residuals avoids forgetting information from the base dataset. Figure 5.4 shows performance of training several ResNet-50's with numbers of residuals from 0 to 6 on the datasets of study. The plots illustrate that worse performance is always achieved when not including any additional residuals in the output block. In addition, appending many residuals yields a decrease in the performance for those datasets more similar to ImageNet (Caltech-101 and Flowers-101), while different datasets (Texture) can benefit from the early coarse representations propagated through the residuals. Given this analysis we can conclude that two residuals are sufficient to achieve good generalization without exponentially increasing the number of parameters.

As with the plots showed in previous sections, Figure 5.4 illustrates that greater generalization may be achieved in mid-stages of the network and not from the bottom (last stage) of the network. The red dots represent the performance at the last residual block ($n = 16$), and they are sometimes outperformed by orange dots, which represent the same metric at residual block 15 ($n - 1$).

5.7.2 Starting stage

The CTL algorithm can potentially be started at any stage of the network. We believe that preserving early coarse representations is crucial to maximize the robustness of the feature vectors. In this experiment we fixed the number of residuals to 2 and used the same training methodology as in section 5.6. We then proceed to execute multiple runs with different starting stages, from stage 3 to the last stage (16).

Without residuals, starting the algorithm at early stages generates a decrease in performance. However, we can help the network avoid losing key coarse representations

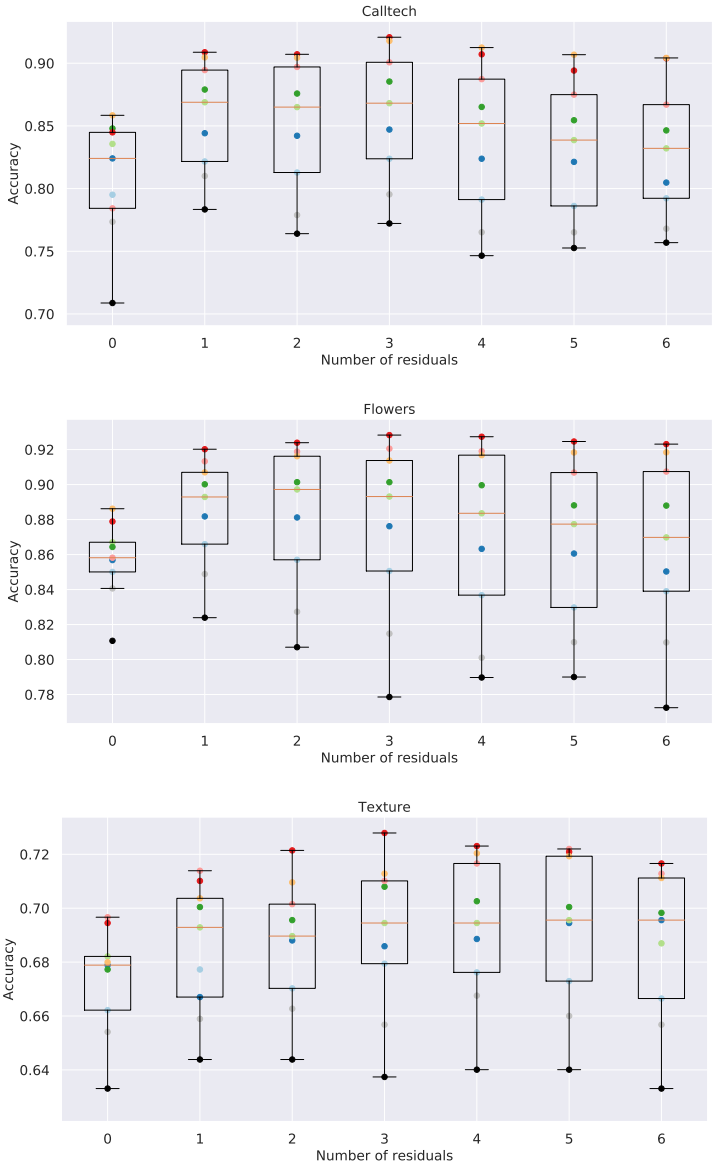

FIGURE 5.4: Performance of CTL at multiple starting stages for three datasets. Each box represents the performance of the network given the number of residuals, the colour dots is peak performance at a particular iteration. Black dots represent iteration zero, red dots last iteration, the rest are mid stages classifiers.

by appending the residual connections. In Figure 5.5 we can see that the performance slightly decreases when starting the algorithm a very late stages regardless of the starting stage. However, since the training cost remains stable across every iteration of the algorithm, starting it early generates more classifiers that can be used to make early predictions without additional memory cost. It can also be spotted that the network progressively builds better generalization, and the best performance is not necessarily achieved at the last stage of the network. This is highly applicable for target datasets that are very different from the base dataset.

Depending on the number of desired classifiers the algorithm can be started at any stage.

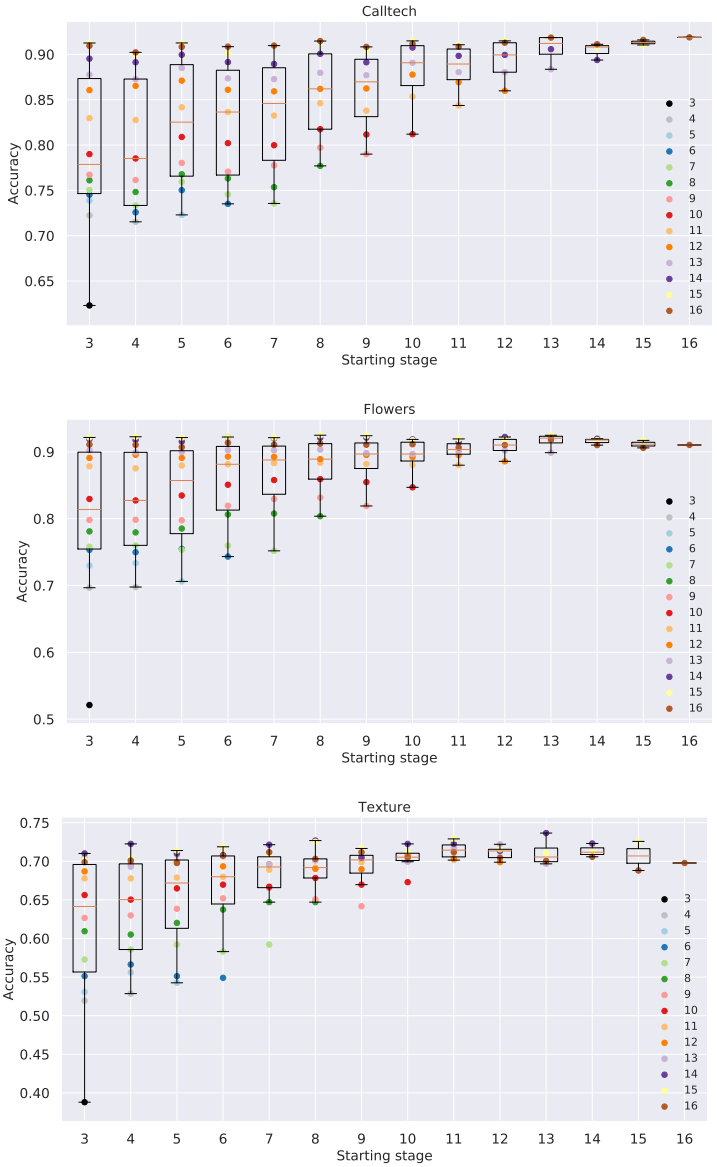

FIGURE 5.5: Performance comparison for multiple CTL runs with different starting stages for three datasets. The parameter was changed from third stage to last stage. Dots represent performance at every classifier of the algorithm.

We have to consider the minimum accuracy required for the transfer learning problem as well as the availability of the hardware.

5.8 Performance versus memory

In this section we evaluate the performance yielded by CTL and fine-tuning in contrast to the required memory for training. We assume a batch size of one for this analysis.

The training memory required for deep networks includes the number of parameters, forward and backward intermediate variables (values and gradients), and the input storage. We did not consider the input storage since it is the same for all the networks, hence, it is a constant that does not affect the comparison. Although, we do consider the dimensionality of the input since it affects the memory required to train the model. Models include a first convolutional layer with no residual connections that also has to be considered in the complexity calculations. For simplicity, we do not include the biases in our calculations.

The memory requirements of a residual network depends on the type of residual block. As mentioned in section 5.5.2, we use two types of residual blocks. The chosen residual block is associated with the architecture.

In Figure 5.6 we show both residual blocks, in any case a convolutional layer is followed by batch normalization. We assume H_{i-1} has dimensions of (C_{i-1}, W, H) (number of input features, and spatial dimensions). A short description of these blocks as follows:

Basic Block. It uses two convolutions with kernel size of 3, the number of feature maps (C_i) is preserved across the residual block. H_i and H_{i-1} must have the same dimensionality to enable the element-wise addition, if this is not the case, H_{i-1} is downsampled using an extra convolutional layer. The shape of the weights of each convolutional layer is then $(C_i, 3, 3, C_i)$. Batch normalization includes two learnable weights (mean and standard deviation) with the same size as the number of channel (C_i). The variables and gradients to save are dependent on the spatial size. In this case the spatial dimensions is preserved, hence, both convolutional layers have the same size for the forward and backwards variables. The shape of the output of both convolutions is then (C_i, W, H), also, the batch normalization includes variables and gradient of the same size as C_i. Equation 5.3 shows the memory complexity of the basic residual block, were P_i represents the number of parameters, and M_i the memory requirements for forward and backward pass.

$$P_i = 2(9C_i^2 + C_i) \qquad M_i = 4(C_i W H + C_i) \tag{5.3}$$

Bottleneck block. This architecture uses three convolutional layers, the 1×1 convolutions are used to increase or decrease the number of feature maps. It has similar properties as the Basic Block, such as H_{i-1} must be equal to H_i and there is a batch normalization operation after every convolution. H_{i-1} contains αC_i feature maps, were α is the expansion factor of the residual block. The weights shape of the 1×1 convolutions are $(C_i, 1, 1, C_i/\alpha)$ and $(\alpha C_i, 1, 1, C_i)$. Similarly, the main convolution holds weights of shape $(C_i, 3, 3, C_i)$. The batch normalization parameters are $2C_i$ for the first two convolutions, and $2\alpha C_i$ for the last convolutional layer. The propagation shape of

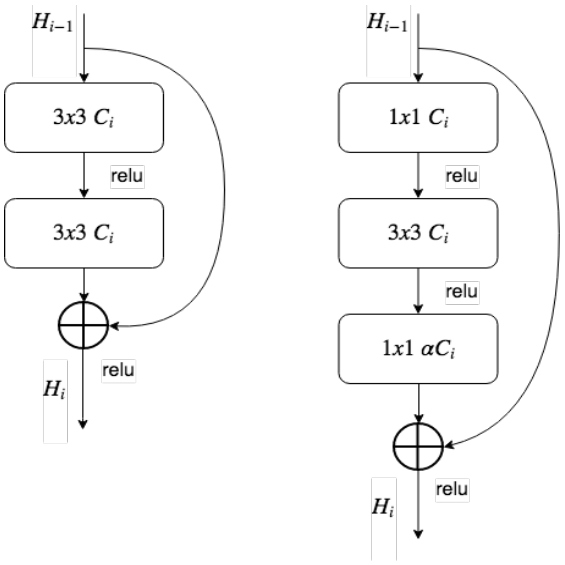

FIGURE 5.6: Residual blocks used in ResNet-34 (left) and ResNet-50/101 (right). H_{i-1} represents the feature maps from the previous residual block, C_i the number of channels of the residual block, H_i output feature maps. Image adapter from the original Residual Networks study [45]

the features across the block is then, (C_i, W, H), (C_i, W, H), and $(\alpha C_i, W, H)$. Equation 5.4 illustrates the parameters and variables complexity of this type of residual block.

$$P_i = 2\alpha C_i^2 + 9C_i^2 + 4C_i + 2\alpha C_i \quad M_i = 4(C_i WH) + 2\alpha C_i WH + 8C_i + 4\alpha C_i \tag{5.4}$$

In order to calculate the whole complexity of the feature extractor, we have to add the complexity of all the residual block $P = \sum_i^n P_i$ and $M = \sum_i^n M_i$. We then have to quantify the memory requirements of the fully connected layers. For the fine-tuning case there is a single fully connected layer. CTL includes additional linear layers but the feature extractor complexity is reduced to $P = P_i$ and $M = M_i$ were i is the stage or iteration of CTL. A given linear layer has $w_l H_{l-1}$ parameters and w_l storage requirements during training, were w_l is the number of units, and H_{l-1} the input vector size.

We quantify the memory requirements of applying transfer learning using CTL and traditional approach. Figure 5.7 compares both approaches in terms of memory and performance. As observed in the figure, blue symbols are shifted to the left with respect with the red ones. Thus, meaning that CTL models only use a fraction of the memory regardless of the depth of the chosen architecture. On the other hand, traditional transfer learning requires more memory as the network gets deeper whilst yielding small improvements over CTL.

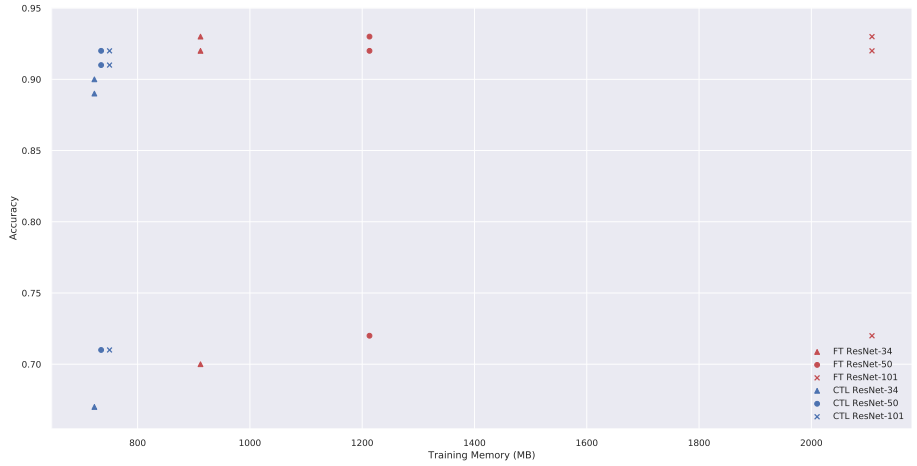

FIGURE 5.7: Memory and performance comparison between Cascade Transfer Learning and fine-tuning transfer learning. Scatter x's represent Caltech dataset, • Flowers dataset, △ texture dataset. CTL yields competitive performance while drastically reducing the memory. We show the maximum memory required during the algorithm, some stages may use less memory. Red symbols are results of fine-tuning, blue symbols CTL.

Potentially, this algorithm can be adapted to make use of three datasets (base, bridge, and target) [62]. Given the memory advantages, one could parallelise the three stages approach and tune each layer from the (1) base dataset to the (2) bridge dataset to the (3) target dataset. This approach could allow training on the target dataset essentially cost free.

5.9 Summary

In this chapter we present a novel transfer learning algorithm using layer wise training. The algorithm aims to adapt early layers to the target domain by using a layer-wise training, which can be executed from mid-stages of the network. By applying this, the pre-trained network smoothly decreases the specificity of the network on the base dataset. This method can achieve competitive performance while using just a fraction of memory. In addition, it can determine the best stage to transfer features to the target dataset. We show empirically that if the target dataset is distant from the base dataset, coarse (early) representations may be more useful as starting point to transfer features.

The algorithm has implementation and execution similarities with Deep Cascade Learning. As discussed in Chapter 4, this layer-wise methodology may be affected by overfitting due to training simpler models iteratively. However, in the case of CTL, overfitting would mean forgetting the feature extractor from the base dataset as opposed to Deep Cascade Learning, where overfitting would lead to greater bias in the model. We believe

preserving the co-adaptation of the layers is more important than fitting the target data at early stages. Hence, underfitted early layers (to the target domain) are acceptable and do not drastically affect the outcome of the algorithm. In this context the term "underfitting" means not forgetting the base representation.

The connectivity pattern of the training algorithms contains residual connections from early stages to avoid losing the hierarchical structure of the deep network. We show competitive results on three different datasets and on three Residual Networks. More importantly, the algorithm drastically decreases the memory required to train the network in comparison to traditional fine-tuning. The influence of the hyperparameters of the algorithm are quantified in section 5.3.

Chapter 6

Cascade Learning for Human Activity Recognition

Deep Learning for Human Activity Recognition (HAR) has shown impressive performance. Several architectures have been proposed to tackle this classification problem. In this Chapter we explore the applicability of Deep Cascade Learning (see Chapter 4 to this sensor problem, as well as its scalability to Temporal Residual Networks. The models used in this Chapter are discussed in later sections.

We consider ResNets and Dilated ResNets for this task. Dilation in the model aims to increase the network's ability to look into the past. We took these networks and applied the Deep Cascade Learning algorithm (see Chapter 4), and directly compared their performance with traditional end-to-end training of the chosen models. The success of these experiments expands the repository of networks on which Deep Cascade Learning can be applied to. We further explore the robustness of the end-to-end models by comparing them with state-of-the-art results, and performing noise and subject-wise cross-validation analyses.

One of the main goals of HAR includes having these networks in small devices (e.g smartphones) to perform inference in real time. Therefore, anticipating the users of certain incidents or monitoring diseases. In most cases, these devices are memory limited and require reduction methods to fit into the device. While cloud computing might be a solution, it can be affected by a poor connection and may generate delays in the inference of the model. Besides inference, this models are often online trained while data is being gathered. Hence, for some applications this models have to be trained in the mobile devices. Thus, Deep Cascade Learning applied to HAR networks can further reduce the training time as well as the resulting model's complexity (using early classifiers). One potential application would be stacking new layers as data is gathered, whilst using the latest fully trained classifier for the required inferences.

The contributions of this chapter include:

- We show that Deep Cascade Learning can be applied to 1D ResNet and 1D Dilated ResNet. Thus, with our methodology models can be trained using a fraction of the memory required to train the end to end network.
- We demonstrate through several experiments that a generic convolutional architecture (Residual Networks referred to as ResNets), outperforms recurrent architectures on this problem. Our results are confirmed on three datasets with greatly varying activities. ResNets consistently performs as well as or better than all other architectures, without including future data.
- We propose the use of ResNet on activities with short duration, and Dilated ResNets on more stationary activities.
- Both architectures show a tolerance to missing values. We evaluated our models with multiple levels of missing data. By using a simple linear interpolation on the missing values, the models preserve their performance when there is a modest amount of missing values in the data.

We discuss deep learning approaches involving Convolutional and Recurrent Networks to classify activities from sensor data. The state-of-the-art includes bi-directional LSTMs, and Deep Convolutional Recurrent Networks. However, in this chapter we emphasise the fact that long term dependencies in activity recognition might not be the best idea, since human activities are mostly periodical and do not hold long term patterns. In addition, we show how Deep Cascade Learning may reduce the memory requirements of the model as well as provide an insight on the depth of the network.

6.1 Human Activity Recognition

Nowadays, smart devices are used on daily basis. These devices often contain several sensors, such as accelerometers and gyroscopes. Recording this data and extracting important information is crucial to solve HAR related problems [31].

Datasets in this field are often captured using smart devices, such as smart phones or smart watches. These recordings contain multiple time series belonging to multiple-sensors (e.g 3D accelerometer, or 3D gyroscopes). Thus, these datasets can be seen as a multivariate time series analysis problem, where the target would be dependent on the dataset labels.

In this chapter we explore a multivariate time series classification problem referred to as HAR. Inputs are the multidimensional sensor data, and the targets contain the activity that is currently been taken by the subject. These activities may vary depending on the

dataset of study, some contain multiple labels (e.g walking, running, standing), while others contain less descriptive binary labels (e.g. Freeze of the gait on subjects suffering from Parkinson's disease).

Early approaches followed a common pipeline: extracting features (e.g Fourier or statistical features); preprocessing those features; and using any standard classification device to find patterns that correlate the features with the class activity [11, 69]. Features were computed on segments of the signals, which were gathered by applying a sliding window. The features were often stacked with their derivatives [69]. Roggen et al. [105] explore the use of Support Vector Machines (SVMs), Random Forest, Gaussian Mixture Models (GMM), and Hidden Markov Models (HMM) as classifiers. These algorithms were then applied to multiple datasets and explored in detail [94, 11].

Two main issues are encountered when applying this pipeline: (a) computing hand-crafted features can be computationally expensive, and may not be applicable to real-time problems; (b) various activities require, in most cases, different features [54], which can become a problem when performing a first weak classification to determine which features to compute.

The HAR community has recently investigated the use of deep learning for classifying activities. One of the first deep learning approaches to HAR used Deep Belief Networks [48] as auto-encoders to compute the features [97]. However, this model was still outperformed by a traditional approach using Principal Component Analysis (PCA) with statistical hand-crafted features. More recently, the HAR community has been applying two types of deep architectures, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Hammerla et al. [41] explore several deep learning networks and provide results of, MLPs, CNNs, and Long Short Term Memory networks (LSTMs), which are a specific type of RNN. All the networks perform relatively well on at least one dataset, although, none of them generalized over multiple types of activities. Ordóñez and Roggen [95] present a deep convolutional recurrent network, which is divided in two stages: a convolutional stage to extract the features from the raw data, and a recurrent stage to learn time patterns across these features. Other recurrent neural network approaches include Ensemble of LSTMs [40], and Binarized Bidirectional LSTMs [26].

Hammerla et al. [41] showed CNNs outperformed LSTMs on the PAMAP2 [102] dataset, which contains more periodic activities were long-term dependencies are not relevant. The DeepConvLSTM approach [95] obtains state-of-the-art results on the Opportunity [104] dataset, and has similar results to bidirectional LSTMs [41, 26].

Recent work has shown impressive results using ResNets on multiple sequence modeling problems [6]. Bai et al. [6] present results on sequence modeling tasks with RNNs and CNNs with dilated convolutions and residual connections; they concluded that CNNs are more effective across diverse sequence modelling tasks. Based on this premise, we

present experiments involving deep residual CNNs to observe their generalization in comparison to LSTMs for HAR tasks.

6.2 Deep Learning for Human Activity Recognition

As reviewed on Chapter 2 of this thesis, Deep Learning has outperformed all other approaches in several core machine learning applications, including computer vision [13, 136], signal processing [64], and time series analysis [127]. The main idea of training deep networks facilitates learning complex features at multiple stages of the network from the raw data. In other words, deep networks learn the features and the classifier simultaneously, as opposed to early approaches where hand-crafted features were computed to then be classified.

HAR is a multivariate time-series classification problem which has been approached using multiple deep learning architectures. We consider deep convolutional networks with residual connections and show that this architecture can outperform all previous proposed models [41, 40, 95]. Particularly, we show that improvements over recurrent models for this sequential task.

We study the case of three datasets, PAMAP2 [102], DaphNet Gait Freeze [4], and Opportunity [104]. We present state-of-the-art results on ResNets and Dilated ResNets, as well as a study on the effect of noise and frequency on these networks. All code used to generate these results can be found at `http://github.com/EnriqueSMarquez/HumanActivityRecognition`.

6.2.1 Long-short Term Memory Networks (LSTM)

We briefly revisit LSTMs in this section, for a fuller picture refer back to Chapter 2.

Ordinary Feed-Forward neural networks are not designed to capture time dependencies. However, Recurrent Networks (RNNs), such as, LSTM networks have shown capabilities to capture these patterns across the time axis [49, 119].

RNNs have been applied on HAR datasets to obtain state-of-the-art results [40, 41]. However, we believe LSTMs are not necessary since activities are typically periodical, hence, there is not much use for long term dependencies. In addition, bidirectional LSTMs have been applied using 'future' data, which limits their usage to offline analysis and no real-time applications [26].

6.2.2 Convolutional Neural Networks (CNNs)

We extensively review CNNs in Chapter 2 for Computer Vision. Most recently, Deep CNNs have been introduced to temporal domains by applying 1D Convolutions, and evaluating these features using non-linear functions. This has been applied in many fields, such as Speech Recognition [111], Speech Enhancement [32], and Speech generation [3].

6.2.2.1 Temporal Convolution

The Temporal Convolution is explained in Chapter 2. In this section we elaborate more about the temporal convolution for sensor data. Given an input multi-sensor (multivariate time series) x with C channels, a biased temporal convolution between x and a set of unidimensional kernels $k_i \; \epsilon \; K_j$ with shape $[H, 1, K_j]$ is defined as:

$$\hat{x}_{t,j} = \sum_{i=0}^{C} \sum_{h=0}^{H} k_i x_{t+h,i} + b_i \tag{6.1}$$

where $\hat{x}_{t,j}$ represents the output feature j at time t, and b_i the bias of kernel k_i. The steps of t will depend on the stride of the convolution.

Figure 6.1 illustrates a temporal convolution with one filter over the multi-sensor signal. Equation 6.1 shows the mathematical operator of convolving a multi-variate signal with the kernel K_j. These kernels are convolved with several time series (multiple sensors) to obtain a set of resulting feature maps. Typically on CNNs, the operation is followed by a non-linear activation function. The nature of the convolutional operation decreases the dimensionality of the signal, hence, inputs are padded to preserve this dimensionality. The kernels (also referred as filters) k_i are trainable and optimized to generate robust and invariant features.

Shallow Temporal CNNs have been applied for HAR on multiple datasets. Moya Rueda et al. [89] present a CNN with 2 convolutional layers with a range of hyperparamters. Similarly, Hammerla et al. [41] applies 2 and 3 layer networks to multiple datasets. Both approaches use max-pooling operations after convolving to downsample the features.

6.2.2.2 Residual Networks

ResNets with 2D Convolutions are explained on Chapter 1. In this section we further discuss these networks. The main difference includes a change in the type of convolution. Instead of using spatial convolutions for images, we use temporal convolutions as explained on section 6.2.2.1.

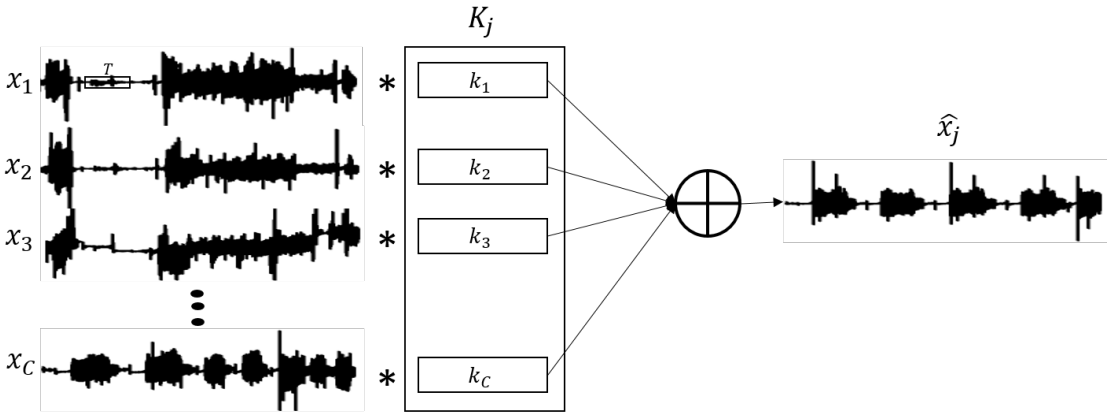

FIGURE 6.1: Temporal Convolution on multivariate time series, window size T and kernel size $[H, 1, i]$. K_j is a multidimensional kernel, which can be seen as multiple 1D kernels. Each kernel is convolved ($*$ sign) with one time series from the input. The resulting individual feature maps are added together to generate a single feature. The output may be further evaluated using an activation function. We illustrate the case of one kernel, typically this process happens several times to generate j feature maps.

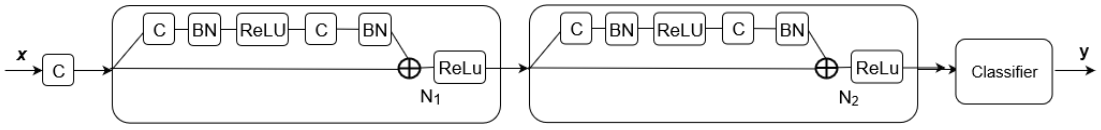

FIGURE 6.2: Residual Network with $N_1 + N_2$ layers, representing first and second resolution of the network respectively. The input x is fed into a convolutional layer C, which is the fed into the Residual Blocks. Each block contains two temporal convolutions (illustrated by C block), two channel-wise Batch Normalization (BN), and ReLU layers (ReLU). The input is added to the output of the ResBlock (circle with plus sign). The features of the last ResBlock is then fed into a classifier, typically a fully connected layer.

State-of-the-art sequence modelling problems using Temporal ResNets include, Time-Series classification [127], and Text classification [19]. Given these improvements in the field using ResNets, we chose to apply these models for HAR.

Figure 6.2 shows a typical ResNet graph with two resolutions. Each box in the diagram illustrates sequentially connected ResBlocks with same resolution. The input is the multi-sensor data, and outputs the activity classification using a fully connected layer. Potentially, these networks can be used to generate a temporal stream of classification with only past data. The first stage contains the same resolution as the input, it is then halved after N_1 ResBlocks. A typical ResNet may contain more than two resolutions.

In contrast with previous CNNs applied to HAR, ResNets do not contain pooling layers. Instead, any downsampling is performed by a strided convolution when required. Dropout is not applied at any stage of the network following the literature of Residual Networks [45, 47].

6.2.2.3 Dilated Networks

A dilated convolution uses the same kernel size as a non-dilated convolution, but the kernel is spaced accordingly to a constant value, which is often referred as degree of dilation (d). By applying temporal dilation to CNNs, the network is capable of finding patterns across longer time steps. Larger dilation correlates with finding longer (time-wise) patterns. In addition, dilated CNNs preserve the dimensionality of the data, which generates no compression losses. Instead of decreasing the resolution, dilated networks increase the dilation (d) of the convolutional layers at multiple stages. Figure 6.3 illustrates the difference between both operators. Residual Networks with dilated convolutions have been proposed in multiple computer vision and speech settings [e.g. 133, 64]. Dilated convolutions in this setting are beneficial to capture longer temporal patters. Thus, enabling better learning of long and not fully periodic activities.

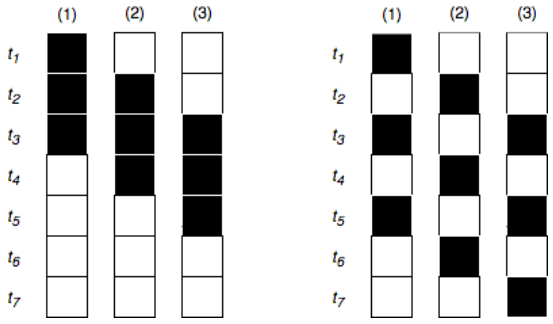

FIGURE 6.3: Illustration of the dynamics of convolution and dilated convolution operators. (Left) Typical Convolution ($d = 1$) (Right) Dilated Convolution ($d = 2$). Vertical axis represent multiple time-steps ($t_1 - t_7$), horizontal axis sequential iterations of the operator ($1 - 3$), black boxes display the receptive field.

6.3 Datasets

We selected three HAR datasets widely used by the community, and with very different classes to see how our model generalize to multiple activities. Each dataset has different motivations and different activities. The protocols for the data acquisition vary for each dataset, thus the sampling frequency, number of subjects and data points vary accordingly. The evaluation was always performed on unseen data, and its evaluation methodology may change across datasets to make results comparable to previous approaches in the field.

All the data used to generate results in this study were downloaded from the UCI repository [23].

6.3.1 Opportunity

The Opportunity dataset [104] contains labeled activities from recordings taken from on-body sensors. The data is collected from 4 subjects using Inertial Measurement Units (IMUs) at 30Hz. Each subject has five files representing different runs. The recordings contain kitchen related activities as labels, such as: *Open Door*, *Open Dishwasher*, and *Close Door*. The fixed evaluation set used in some experiments was selected using the protocol in [89] [95]. The resulting dataset contains 79 dimensions when concatenating all the sensor data, corresponding to 79 one dimensional time series of the 79 sensors. For most experiments we used run 2 from subject 1 as validation set, and runs 4 and 5 from subject 2 and 3 as test set as per the related literature [41, 26, 40].

6.3.2 PAMAP2

Similarly to the Opportunity dataset, PAMAP2 [102] consists of labelled activities among nine participants. Each subject carried out twelve prolonged activities daily activities such as standing and walking, and more complex activities such as Nordic Walking and Vacuum Cleaning. The sensors used to collect the data were IMUs containing accelerometers, gyroscopes, and magnetometers. In addition to the IMUs, the data also contains heart rate and temperature recordings. The data was collected at 100Hz with a total of 52 dimensions. We excluded subject nine due to the reduced recording time in comparison with the rest of the participants. The fixed evaluation set was selected following the protocol in Moya Rueda et al. [89] and Hammerla et al. [41], which uses subjects 5 and 6, as validation and test sets respectively.

6.3.3 Daphnet Gait

The Daphnet Gait Freeze dataset [4] is as binary classification problem. The data was recorded on ten subjects diagnosed with Parkinson's Disease, and was labelled accordingly to whether the subject was suffering from "Freeze of the gait incident". The labels are *Freeze* and *not Freeze* . The subjects performed activities with high likelihood of inducing freezing of gait (a motor complication in initiate movements). The sensors were placed on the ankle, above the knee and on the chest. The data was recorded at 60Hz and contains a total of 9 dimensions. We evaluated this dataset using subject 9 as validation set, and subject 2 as our test set [41].

6.4 Experimental setup

We test 1D Dilated and non-Dilated ResNets on the datasets presented in section 6.3. In our experiments we present results on fixed evaluation sets chosen from the literature

to compare our models with the state-of-the-art. The second experiment includes performing cross-validation across subjects (leave-one-person-out [126]) to further compare both models. We also test our models on downsampled data to visualize the effect of the sampling frequency on the performance.

Both architectures have the same structure, three downsampling/change in dilation, with 3, 4, 4, 5 ResBlocks for the first, second, third and forth resolution respectively. Each ResBlock consist of 2 convolutional layers, hence, the models contain 33 layers including the classification stage (MLP). We tested multiple ResBlocks arrays, such as Pre-activation and Bottleneck, but found no improvements over the Basic Block. The first convolution has 64 filters, and is duplicated after every downsampling/change in dilation. The kernel size was empirically tuned to 15, and inputs to the convolutions were always padded to preserve dimensionality. The output layer contains one fully connected layer with softmax activation, and the number of output units depend on the dataset. All the networks were trained for 400 epochs, using Stochastic Gradient Descent with a learning rate schedule, momentum of 0.9, and cross entropy loss. The learning rate starts at a high value to 'warm up' the network (0.1) and is dropped by a factor of 10 on epochs 3, 100 and 200. The network is regularized using weight decay with a value of $10e-4$. The input shape to the network is (B, T, L) were B represents the batch size set to 32, T the number of channels (dimensions), and L the window size (length of the sequence).

The evaluation was performed following [41]. In order to make the results comparable with the literature, we used weighted f_1 score (Eq. 6.2) for the Opportunity dataset. For the remaining two datasets we used mean f_1 score (Eq. 6.3). In both equations, 6.2 and 6.3, $prec_c$, $recall_c$, and N_c represent the precision, recall and number of data points of a given class c. N_{total} is the number of samples in the dataset.

$$F_w = 2 \sum_c \frac{N_c}{N_{total}} \frac{prec_c \times recall_c}{prec_c + recall_c} \quad (6.2) \qquad F_m = \frac{2}{\|c\|} \sum_c \frac{prec_c \times recall_c}{prec_c + recall_c} \quad (6.3)$$

6.5 Results

In this section we discuss and execute some experiments. A short description of these experiments is presented as follows:

- Deep Cascade Learning. We explore the applicability of Deep Cascade Learning on 1D ResNets using sensor data. In addition, we quantify and compare the memory reduction obtained by applying our algorithm to these models.

- Comparison with state-of-the-art. We compare both ResNets with current state-of-the-art models. In addition, we quantify the performance variance and show mean, standard deviation and best across 20 training runs.
- Subject wise cross-validation. This experiment provides a better insight on how well these models generalize across multiple subjects. It is particularly interesting given the nature of the applications of HAR, one would be interested in a machine that can be applied to any subject.
- Sampling Frequency effect on Temporal ResNets. We test how sensor data with different frequency may affect these residual models. This is performed by down-sampling the data several times and see if the models can still capture meaningful representations.
- Noise tolerance. Sensor data might be affected by failures in the hardware. These failures are often presented as missing values. Hence, this experiment enable us to observe if a simple interpolation of the data can be used to append missing values while not drastically losing performance of the models.

6.5.1 Cascading ResNets for HAR

In this section we evaluate the performance of Deep Cascade Learning applied to these models. In Chapter 4 we present an algorithm for layer-wise training on 2D CNNs. Similarly, we quantify the performance of applying such algorithm to 1D ResNets and 1D Dilated ResNets. The success of this experiment suggests that Deep Cascade Learning can scale to 1D CNNs and potentially to any type of differentiable neural network.

Table 6.1 shows results of applying cascade learning on both ResNet architectures on the test sets for the datasets of study. To maximize the performance, these networks are better trained in an end to end fashion, however, when cascading it yields competitive results against the state of the art (Table 6.2). When cascading these deep networks, the performance is very stable after approximately residual block seven (it may vary depending on the run). We know Deep Cascade Learning can be affected by overfitting if not tuned carefully. Moreover, cascading is not fully taking advantage of deep representations while still providing competitive results. We believe Deep Cascade Learning is more affected by imbalance datasets and lack of data, given that these data properties also induce overfitting. A biased dataset can lead to a model being more likely to make the same prediction, which can be seen as overfitting. As explored in Chapter 4, early iterations of Cascade Learning might overfit. Thus, we applied greater weight decay ($10e-3$) and just allowed each submodel to see the data five times. Even by applying this regularization scheme, the models still fit the data on mid stages of the network, and performance does not improve on deeper iterations of the training algorithm. We did not fully tune the Cascaded model, further improvements might be achieved by better

	PAMAP2	**DaphNet**	**Opportunity**
ResNet	0.90 (0.91) ± 0.01	0.75 (0.83) ± 0.04	0.91 (0.91) ± 0.00
Dilated ResNet	0.90 (0.91) ± 0.02	0.72 (0.77) ± 0.03	0.91 (0.91) ± 0.01

TABLE 6.1: Results on Cascading ResNets and Dilated ResNets on HAR data. It reads *mean* (*best*) ± *std* across 10 runs using the fixed training, validation, and test datasets provided by the literature.

regularizing and tuning this algorithm the HAR task. Most hyperparameters stayed the same to provide a fair comparison against their end to end counterparts.

Memory Complexity. In order to quantify the memory complexity advantages of cascaded 1D CNNs, we calculate the end to end and cascade memory requirements for training. The calculations remain similar as in Chapter 5 Section 5.8. The main difference is that kernels are 1D, features are 2D (width, channels), and dilated networks do not perform downsampling. For each dataset, calculations may vary due to different shape in the data (window size, number of sensors, number of labels). We show average memory across all datasets to simplify the results. The end to end training requires approximately 1.55 Gb of space to train a standard ResNet with the hyperparameters specified in Section 6.5.2. While cascading only requires 0.25 Gb, which represents a memory reduction of ~ 6 times. On the other hand, dilated ResNets use slightly more memory in both cases, 1.60 Gb and 0.27 Gb for end to end and cascading respectively. Notice that for cascaded networks we show space and memory complexity for the iterations with greater cost.

6.5.2 Comparison with state-of-the-art

In this section we compare Temporal ResNets with previous deep learning approaches on the datasets of study. Specifically, we directly compare with the following architectures:

- Ensemble LSTMs [40]: merges multiple LSTMs with a fused training algorithm to generate a single system. Averaging networks with similar methodologies have always shown improvements.
- DeepConvLSTM [95]: combines a CNN to generate abstract features with an LSTM to find time patterns on these features.
- Binarized-BLSTM [26]: binarization of a bi-directional LSTM. Binarizing the weights and activations compresses the network to fit into devices easier.
- CNN [41]: shallow three layer CNN with max pooling operators and hyperbolic tangent activation.
- LSTM-F and LSTM-S [41]: recurrent networks where the time steps of the input data is either concatenated (model F), or fed as a sequence (model S).

- b-LSTM-S [41]: bidirectional sequence LSTM.

In Table 6.2 the results on each dataset for both architectures are presented as average, best, and standard deviation of the test f_1, *mean (best)* $\pm$ *std*.

Overall, our models generalize better if consider the mean performance of all three datasets. By having good initialization, the models can outperform LSTMs by 4 and 5 % without including any future data.

6.5.2.1 PAMAP2

ResNets on average perform similarly to previously published networks. Its best validation f_1 performs slightly better than the CNN trained on [41]. It also has a low standard deviation which suggests that the network is not affected by the randomness of the training algorithm. The Dilated ResNet can potentially perform better than any other network. This is due to the activity types of the PAMAP2 dataset. Dilated ResNets are better are classifying prolonged activities. Figure 6.4 shows how the ResNet misclassifies *sitting* with *standing* more times than the Dilated ResNet.

We can observe in Table 6.2 that both architectures perform better than the CNN and LSTM-F in Hammerla et al. [41]. The best results can be obtained using Dilated ResNet. The Binarized-BLSTM uses future data and does not necessarily perform better, which gives an advantage to our models when it comes to applications and deployment.

6.5.2.2 DaphNet

Both models can indeed outperform LSTMs for these periodic movements, and on average the ResNet performs 1% better than LSTM-S. The variance of the Dilated ResNet is high, but it can converge into a network with very high f_1 score (0.82) in comparison with the literature. For this dataset, ResNets are more likely to provide better and stable results.

6.5.2.3 Opportunity

Due to the short activities presented in the Opportunity dataset, Dilated ResNets do not perform as well as ResNets, however, they still provide competitive results. ResNets on the other hand are more suitable and generate better results than b-LSTM-S by 1.3% at best. The models are almost invariant to poor initialization and randomness on this dataset. They obtain a variance of 0.0075 on average.

We downsampled the signals to approximately 30 Hz and segmented them using a sliding window approach. The size of the segments are 5.12 sec, 1.5 sec, and 1.5 sec for PAMAP2,

	PAMAP2	**Daphnet**	**Opportunity**	Mean	Std
Metric	F_m	F_m	F_w	-	-
Ensemble LSTMs [40]	0.854	-	-	-	-
DeepConvLSTM [95]	-	-	0.917	-	-
Binarized-BLSTM [26]	0.93	-	-	-	-
CNN [41]	0.937	0.684	0.894	0.84	0.14
LSTM-F [41]	0.929	0.673	0.908	0.84	0.14
LSTM-S [41]	0.882	0.76	0.912	0.85	0.08
b-LSTM-S [41]	0.868	0.741	0.927	0.85	0.10
ResNet	0.93 (0.94) ± 0.01	0.77 (**0.83**) ± 0.03	0.92 (**0.94**) ± 0.008	0.87 (**0.90**)	0.08
Dilated ResNet	0.92 (**0.95**) ± 0.02	0.73 (0.82) ± 0.10	0.90 (0.91) ± 0.007	0.85 (0.89)	**0.06**

TABLE 6.2: Comparison with state-of-the-art for ResNet and Dilated ResNet using end to end training. The results were generated using 20 runs and by selecting the best validation f_1 as stopping criteria. The notation is *mean (best)* ± *std* across all the runs.

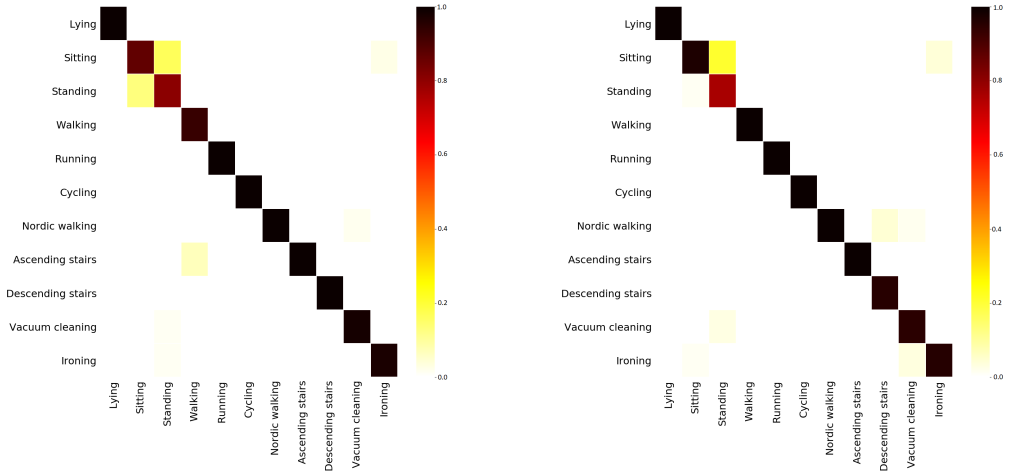

FIGURE 6.4: Normalized confusion matrices for the PAMAP2 dataset for (left) ResNet (right) Dilated ResNet. ResNet is more likely to misclassify sitting with standing, while the Dilated ResNet mostly misclassifies vacuum cleaning and sitting with ironing.

DaphNet, and Opportunity respectively, and sampled with an overlap of 50%. The networks were run 20 times to measure convergence and effect of initialization, and were stopped when the validation f_1 plateaued or dropped.

6.5.3 Subject wise cross-validation

We further compare our models by cross-validating across subjects. We aim to see how well these models generalize among subjects, as well as maximizing the amount of data by using the default frequency.

Table 6.3 shows results leaving one person out as validation set. The ResNet performs 1.4% better if we consider the mean difference between datasets. The dilated ResNet has a higher standard deviation, it generalizes worse across multiple subjects, although, still providing competitive results.

	Dilated ResNet			ResNet		
Subject	**PAMAP2** (F_m)	**Daphnet** (F_m)	**Opp** (F_w)	**PAMAP2** (F_m)	**Daphnet** (F_m)	**Opp** (F_w)
1	0.92	0.72	0.84	0.93	0.72	0.87
2	0.89	0.72	0.78	0.92	0.75	0.90
3	0.95	0.72	0.77	0.95	0.74	0.76
4	0.95	1.0	0.82	0.96	0.99	0.82
5	0.93	0.66		0.95	0.63	
6	0.92	0.64		0.95	0.65	
7	0.98	0.69		0.98	0.66	
8	0.72	0.65		0.64	0.67	
9		0.78			0.78	
10		1.0			1.0	
Mean	0.933	0.758	0.803	0.944	0.759	0.835
Std	0.03	0.134	0.03	0.02	0.134	0.059

TABLE 6.3: Subject cross-validation results for all three datasets using the Dilated ResNet and ResNet models. The values are maximum validation accuracy achieved during training leaving one person out of the training data.

6.5.4 Sampling Frequency effect on Temporal ResNets

Temporal Dilated ResNets emulate looking at the data at multiple frequencies, for example when $d = 2$, the network looks at the signal with half its frequency. Hence, we execute some experiments to see how the speed of the sensors may affect these networks. For this experiment, we only use the PAMAP2 dataset since it contains a higher frequency than the rest and can be downsampled several times. We have to also take into account that, downsampling the data generates fewer samples to train the network and this may affect the generalization performance. The downsampling was done by linearly skipping some values in the data. Each network was run four times to measure its invariance. The window size was always fixed to 5.2 seconds. However, we did not increase the number of parameters of the network. We anticipate linear results if the ratio $\frac{kernel\ size}{number\ of\ samples}$ is preserved.

Figure 6.5 shows the performance of our networks on multiple frequencies for the PAMAP2 dataset. When drastically downsampling the data, dilated convolutions can generalize better. This is seen in the performance between 10 Hz to 17 Hz. In any other case, the ResNet seems to be performing best. The variance of the ResNet is doubled on the dilated version. It also explains why the best results may be generated using dilation.

The performance drops at high frequencies even though at this stage there is more data available. This negative slope in Figure 6.5 can be rectified by increasing the kernel size, which will increase the receptive field and look at more samples at a given time.

6.5.5 Noise tolerance

For this experiment we used same metrics and networks to those trained for Section 6.5.2 results. Ponce et al. [98] studies a methodology to measure noise robustness of

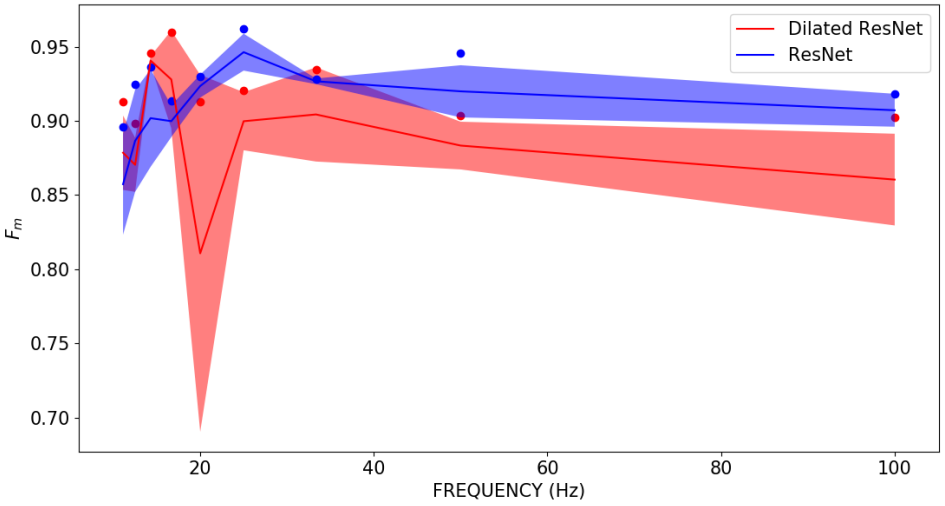

FIGURE 6.5: Performance on test data of PAMAP2 at multiple frequencies on four runs each. The shade of the curves represent the variance of the models and the scatter dots the best achieved performance.

machine learning algorithms for HAR datasets. We simulated random missing values on the test data on the range 1-99 % of missing values. We chose this scheme due to its likelihood in HAR applications (failing of sensors lead to missing data points), and it is considered a noise insertion method.

Before feeding the inputs to the network, the missing values were imputed using a simple linear interpolation. In our experiments we computed the performance of 20 networks per dataset, and obtained values for mean, standard deviation and best performance. The networks were trained on the default clean data.

Figure 6.6 illustrates the performance of the networks for the given range of missing values. The networks trained on PAMAP2 and DaphNet are more affected by noise given than these datasets contain downsampled prolonged activities. On these two datasets it is more likely to randomly add missing values to important features, while in the Opportunity dataset, due to its imbalance, most of the inputs are false positives that can be easily approximated with the interpolation scheme. The dilated model approximately doubles the variance across all the datasets of the ResNet with a mean value of 0.089. When initial conditions are best, both models perform similarly given the noise range. Due to simplicity, we did not include the variance of the models in Figure 6.6.

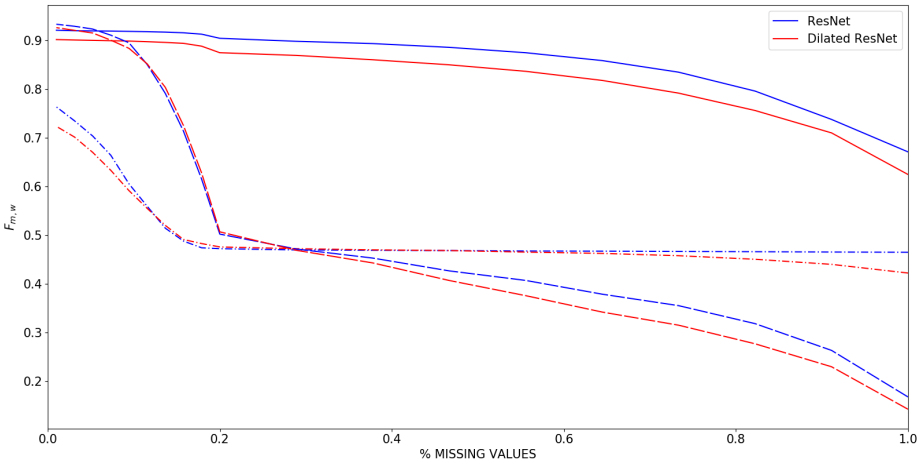

FIGURE 6.6: Missing values tolerance for both models on three datasets. Red curves represent mean performance for Dilated ResNet, blue curves ResNet. Opportunity (solid), PAMAP2 (dashed), DaphNet (dashed and dotted).

6.5.6 Discussion

In section 6.5 we illustrate the performance of ResNets and Dilated ResNets in comparison with other architectures, especially LSTMs. There are several advantages of our models over the literature: (a) Higher f_1 score on the tested datasets, (b) CNNs are easier to interpret since the weights and features can be visualized for multiple activities, (c) Binarization of LSTMs is a way of compressing the network weights, however, CNNs can be not only binarized but also pruned [42] to potentially fit into small devices (e.g smartphones, Rasberry pis), (d) some models in the literature are bidirectional and uses future data, while CNNs are trained only on the past sequence, (e) our models do not use pooling or dropout in contrast with the literature, making the feature extractor fully convolutional and potentially transferable among multiple datasets (g) CNNs in the literature are shallow, using deeper CNNs can provide better results, this is only possible due to the residual connections, (h) data augmentation can potentially be applied to CNNs to learn more invariant features, while this is not the case for LSTMs where data augmentation may generate noise in the model, (i) ResNets can be used to generate a stream of classification by inputting an entire sequence (without segmenting the signal), this is particularly useful in real-time applications.

Both architectures outperform the literature in most cases. However, whether to choose a dilated or a typical ResNet will depend on the dataset. The dilated ResNet is affected by poor initialization, while its counterpart is more invariant to this randomness. The model size is the same in both cases, although, the cost of propagating the inputs is higher for the dilated ResNet due to necessary increase in the padding.

When applying these models to high frequency data, we have seen that it is necessary to increase the kernel size of the network. By doing this, the memory required to train the model may increase linearly. Hence, in current work we are exploring the use of deep cascade learning, a novel architecture suitable for memory efficient training and transfer learning [82], on HAR.

6.6 Summary

We have Residual Networks for HAR, motivated by speech and computer vision models and recent work on sequence modelling. The architecture ResNet has been applied to many computer vision tasks, and the dilated version to speech tasks. We compare both dilated and not dilated ResNets with state-of-the-art results on the datasets PAMAP2, DaphNet Gait Freeze, and Opportunity. Our method performs the best in certain cases, and provides competitive results on average. We argue that LSTMs are more computational extensive and less robust for these sequence modelling tasks. Especially, when applying to datasets similar to PAMAP2, with mostly low term dependencies such as sports, or gait analysis. We illustrate the benefits of downsampling the data to match the kernel size. In addition, we show that these networks can tolerate certain level of missing values in the data, however, further work includes improving this tolerance through augmentation.

Additionally, we explore applying Deep Cascade Learning on 1D ResNets and 1D Dilated ResNets. We quantify the reduction of memory requirements by applying our layer-wise algorithm. More importantly, given the imbalance, the nature of the datasets, and the number of hyperparameters, empirical tuning can become computationally extensive. Hence, DCL can provide insights on depth and number of required iterations to find a better minimum. Cascaded ResNets also take advantage from less convergence variance, and thus, have less likelihood for converging into a poor minimum. Finally, when these cascaded networks are stored in any smart device, early classifiers can be used to make early predictions and avoid the need of continuing the forward propagation, ultimately providing faster inference given a performance threshold. However, in terms of performance, end to end training seems to obtain better results than DCL when applied to ResNets.

Further studies include, testing dilation on the width axis of the network while keeping the residual connection, reflected as an inception module [120] with multiple dilations instead of kernel sizes. The datasets used on this manuscript did not contain high heterogeneity or proportion of missing data/features. To further measure its robustness is ideal to compare it using the Heterogeneity Human Activity Recognition (HHAR) dataset, which records data with different devices holding more similarities to the data gathered for applications. However, models applied to this dataset, such as Deep Sense

[131] are not directly comparable since it uses Fourier features (following Stisen et al. [118]) as input to the CNN. In addition, our models have shown learning transferability properties in multiple deep learning fields.

Chapter 7

Conclusions & Future Work

This work has demonstrated that by constructing a cascade architecture which trains a neural network in a layer-wise fashion, we can achieve significant reduction in computational complexity at the expense of small decrease in performance. We have shown this on several benchmark problems related to computer vision, signal processing, and transfer learning applied to computer vision. Thus, Deep Cascade Learning enables the users to train very deep networks with no additional hardware requirements. A short description of each chapter contributions and findings are presented as follows:

Chapter 2 & 3 reviews deep learning for both computer vision and signal processing. We review the literature from early approaches on learning deep representations to state-of-the-art architectures and algorithms. Specifically, we discuss deep learning from early MLPs and perceptrons to the main breakthrough of AlexNet on training very deep networks on large scale datasets. This was possible due to advancements in hardware and the availability of the data. We then explore the literature on layer wise training and most successful attempts to iteratively and hierarchically train these networks. Most importantly, we focus on Fahlman's Cascade Correlation algorithm, which was the first unit wise training algorithm. Surprisingly our review discovers that, the Cascade Correlation algorithm shares many properties with the state-of-the-art methodology, such as dropout and residual connections.

Chapter 4 presents a greddy supervised layer wise training algorithm, which we named Deep Cascade Learning (DCL). The algorithm was successfully tested on two benchmark image classification datasets with 10 and 100 classes. The models used in this chapter include VGG and The All CNN networks. DCL is supervised and provides time and memory complexity reduction, which is due to a decrease in the overall number of epochs with no additional propagation through already trained layers. This method is inspired by and shares similar properties with Fahlman's Cascade Correlation. However, it is applied to a different scale to modern deep learning architectures. An additional motivation is the fact that DCL directly tackles the vanishing gradient problem. We

quantify the effect of vanishing gradient problem by computing the magnitude of the gradients at every stage of the network. Our findings are corroborated by several experiments and analysis which directly compares Deep Cascade Learning with traditional end to end training: (a) Memory complexity comparison, including discussions on optimal and naive implementations; (b) Quantifying time complexity for both algorithms; (c) Feature convergence, showing that Cascade Learning is more likely to converge to the same minimum, which decreases the impact of poor initialisation; (d) Magnitude of gradients analysis, to observe if there is indeed a increase in the magnitude of the gradients at early stages; (e) Performance improvements when initialising with Cascade Learning and Fine-Tuning the whole network.

Chapter 5 presents an extension of Deep Cascade Learning algorithm applied to transfer learning, which we refer to as Cascade Transfer learning (CTL). We explore this transfer learning methodology on three target datasets and used pre-trained Residual Networks on ImageNet. CTL uses the same implementation as Cascade Learning and shares similar advantages. Very deep pre-trained networks can be computational expensive, and it may require more than one device to fine tune the network. Our approach allows us to train these models without additional hardware. Additionally, the algorithm is capable of determining at which stage of the network the features are better transferred to the target dataset. The early classifiers can be stored and used to make predictions on 'easy' inputs, avoiding the need to continue propagating through the network. We list the experiments performed on this Chapter as follows: (a) Comparison with state-of-the-art transfer learning: we compare against Fine-Tuning transfer learning, which is known to be a very effective approach to transfer features; (b) The algorithm contains two hyperparameters: we quantify the effect the number of residuals and starting stage concluding that two residuals and starting at mid-stage of the network is sufficient to properly cascade the networks of study; (c) Memory complexity analysis: we show that our algorithm requires just a fraction of the memory while achieving competitive performance against fine-tuning transfer learning. The results of these experiments suggest that CTL can be executed with less complexity requirements than other transfer learning approaches.

Chapter 6 explores deep learning models for signal processing. In particular, we test the applicability of Deep Cascade Learning on these temporal models. Human Activity Recognition (HAR) uses wearable sensor data to classify activities. We show how state-of-the-art deep learning architectures can yield better performance on this multivariate time series classification problem. In this chapter we demonstrated two main aspects of Deep Cascade Learning. Firstly, networks with branches such as residual networks, can also be cascaded without drastically affecting the performance. Secondly, DCL can be applied to signal processing models including 1D convolutions while preserving generalization of deep features. These models also yield similar advantages to cascading 2D CNNs. The experiments involve ResNets and Dilated ResNets on three benchmark

datasets with a wide range of different activities. These experiments include: (a) Comparison of our models with state-of-the-art results; (b) Subject wise cross-validation as a better metric for this problem; (c) Experiments to measure tolerance of networks to frequency and missing values; (d) Performance comparisons between 1D ResNets and 1D Cascaded ResNets for HAR.

In summary, this thesis explores the literature on Deep Learning focused on computer vision and signal processing. In addition, we developed Deep Cascade Learning, a novel layer-wise algorithm training deep networks. This algorithm is more memory efficient than traditional end-to-end training, and can potentially allow to train endlessly deep networks. Furthermore, we developed Cascade Transfer Learning, which takes Deep Cascade Learning a step further by applying a similar algorithm to pre-trained models. In addition, we quantify the performance of Deep Cascade Learning on a multi-sensor classification problem (Human Activity Recognition). We conclude that Deep Cascade Learning provides an alternative to traditional end-to-end training, and can further enable the community to train deep models on small devices. Even though the performance of DCL sometimes does not match its traditional end to end counterpart, it still provides complexity advantages that can be beneficial for applications and deployment of machine learning models. Our developed algorithms have only been tested in the computer vision and signal processing field. We encourage the community to try our methods in other machine learning fields.

Future work

Layer wise training of deep networks offer several advantages. There are still many ideas to explore around this topic that are applicable to both Deep Cascade Learning and Cascade Transfer Learning.

The Deep Cascade Learning algorithm requires fine-tuning to match the performance of end to end. Our key ideas for further exploration on Deep Cascade Learning are the following:

- One of the issues of our layer-wise methods is overfitting at early layers. Early stages of the network must hold coarse representations to be able to learn finer representations at deeper layers. To avoid overfitting, we reduce the number of times each sub-model sees the data. This is a naive approach that yields good results. However, further regularization can be implemented to prevent fitting the training data. From our point of view, several approaches can be taken to alleviate overfitting at early stages. One could prune after every cascade iteration and induce regularization by avoiding redundancy of the learnt features, which ultimately reduces noise at mid stages of the network. Similarly, binarization [100]

of already cascaded layer can further regularize the algorithm. Another way of regularizing would be by normalizing the features after cascading to decrease the amount by what the hidden features shift around. Since the layer is frozen, we can propagate all our input data and find normalization parameters after every iteration. By doing this, we decrease the covariance and avoid specificity of the features.

- On our layer-wise methods, we have to tune the number of epochs on which the algorithm is started. Hence, the stopping criteria depends on either the number of epochs or performance on validation set. We believe a better criteria would take into consideration how much information has been compressed at each layer, and give us an insight of whether an improvement can be achieved by appending a new layer. An approach might potentially include the use of information bottleneck theory [113]. The algorithm could automatically anticipate when each iteration should be stopped by maximize the correlation between inputs and outputs. Doing these analyses can also give us better understanding of DCL and CTL.

- It is worth exploring the limits of depth when applying Deep Cascade Learning. Knowing that DCL gets less complex as the algorithm progresses, it could potentially be used to train an 'infinite' layer network. The result would be an adaptive architecture that self-tunes certain hyperparameters (e.g. filters width/dimensions, depth, network width) and progressively learns better representations. The programmer would then be relying less on empirical tuning. The obvious hyperparameter to automate when using Deep Cascade Learning would be the depth of the network, but the principle can also be considered to also learn number of filters and shape of kernels. In addition, due to the memory reduction, cascade learning can train wider networks with more number of filters.

- There are some ideas that can make DCL more similar to Cascade Correlation. Exploring these ideas might enable better learning of our layer-wise algorithm. These ideas include fully connecting the outputs of Cascade Learning, and applying dropout on frozen layers. More importantly, Cascade Learning could adapt the idea of evaluating candidate layers and selecting the most suitable one. Since there is a decrease in the memory usage, training multiple layers can be parallelised in a single or multiple devices, and essentially create the 'pool' of candidates for free (time-wise). This would follow both Cascade Correlation [28] and AdaNet [20].

- Classifiers on early stages learn different representations than those on later stages. Therefore, after finishing the iterations of our layer-wise models, one could explore methods to combine these classifiers. For example, using a ensemble model to merge the classifiers, or a voting system to combine the predictions, or boosting. These could lead to a bigger network with merged representations.

- The idea of cascading networks could be scaled to other domains. Specifically, we are interested on whether Recurrent Networks can be cascaded in a similar way as CNNs. It is not as straightforward as CNNs since the structure of LSTMs is not hierarchical, and is dependent on time patterns rather than spatial patterns. Furthermore, exploring the cascaded features of MLPs is also of interest. The motivation of only using CNNs on these work comes from the nature of these networks and its behaviour as feature extractors.

- One interesting idea to explore would be Deep Cascade Learning as online adaptive model. Since cascaded networks are trained layer by layer, one could adaptively increase the number of layers as new data is coming. Already learnt classifiers can be used until new layers have been evaluated, thus, appending to the network new representations learnt on the 'recently' obtained data. This property has numerous advantages for deployment and execution of live models. So far, the community addresses this issue by training the network from scratch when is necessary, which would be more inefficient than our proposed cascaded approach.

- Recent implementations of CTL show that adding residual connections to already pre-trained nets can boost the co-adaptation of the layers. Hence, we could avoid the need of fine tuning after cascading to reach a better performance. Exploring this can further reduce the probability of overfitting at early stages. Ultimately, this could lead to an enhance of Deep Cascade Learning and further reduce the required memory to match performance against end to end.

- DCL networks learn better representations at every stage. Therefore, using these networks as transfer learning devices might be beneficial to learn better representations of the target domain. These early representations might be more correlated with the target domain than features extracted from an end to end trained network.

- For HAR, it is worth of exploring if the concept of DCL can be scaled to more WaveNet [64] like networks with oscillating dilations and multiple output blocks. This analysis may also include quantifying the performance of DCL given multi-activation CNNs.

- Finally, further analysing of DCL by applying it on other fields, such as Speech and Natural Language Processing, can lead to a better understanding of DCL and can scale up the repositories on which DCL is applicable to.

References

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE, 2012.

[2] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[3] Sercan Ö Arık, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Andrew Ng, Jonathan Raiman, et al. Deep voice: Real-time neural text-to-speech. In *International Conference on Machine Learning*, pages 195–204, 2017.

[4] Marc Bachlin, Meir Plotnik, Daniel Roggen, Inbal Maidan, Jeffrey M Hausdorff, Nir Giladi, and Gerhard Troster. Wearable assistant for parkinsons disease patients with the freezing of gait symptom. *IEEE Transactions on Information Technology in Biomedicine*, 14(2):436–446, 2010.

[5] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[6] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[7] Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

[8] Gustavo EAPA Batista, Xiaoyue Wang, and Eamonn J Keogh. A complexity-invariant distance measure for time series. In *Proceedings of the 2011 SIAM international conference on data mining*, pages 699–710. SIAM, 2011.

[9] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 153–160. MIT Press, 2006.

[10] Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[11] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, 46(3):33, 2014.

[12] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[13] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[14] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[15] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[16] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[17] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[18] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUS). *arXiv preprint arXiv:1511.07289*, 2015.

[19] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *European Chapter of the Association for Computational Linguistics EACL'17*, 2017.

[20] Corinna Cortes, Xavi Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. *arXiv preprint arXiv:1607.01097*, 2016.

[21] Marc-Alexandre Côté and Hugo Larochelle. An infinite restricted boltzmann machine. *Neural computation*, 28:1265–1288, 2016.

[22] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd national conference on Artificial intelligence-Volume 1*, pages 540–545. AAAI Press, 2007.

[23] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[24] Jose Dolz, Xiaopan Xu, Jerome Rony, Jing Yuan, Yang Liu, Eric Granger, Christian Desrosiers, Xi Zhang, Ismail Ben Ayed, and Hongbing Lu. Multi-region segmentation of bladder cancer structures in MRI with progressive dilated convolutional networks. *arXiv preprint arXiv:1805.10720*, 2018.

[25] Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification 2nd Edition*. John Wiley & Sons, 2012.

[26] Marcus Edel and Enrico Köppe. Binarized-blstm-rnn based human activity recognition. In *Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on*, pages 1–7. IEEE, 2016.

[27] Scott E Fahlman et al. An empirical study of learning speed in back-propagation networks. 1988.

[28] Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pages 524–532, 1990.

[29] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

[30] Christoph Feichtenhofer, Axel Pinz, and Richard P Wildes. Temporal residual networks for dynamic scene recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4728–4737, 2017.

[31] Thomas Fritz, Elaine M Huang, Gail C Murphy, and Thomas Zimmermann. Persuasive technology in the real world: a study of long-term use of activity sensing devices for fitness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 487–496. ACM, 2014.

[32] Szu-Wei Fu, Yu Tsao, and Xugang Lu. Snr-aware convolutional neural network modeling for speech enhancement. In *Interspeech*, pages 3768–3772, 2016.

[33] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980. ISSN 1432-0770.

[34] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[35] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[36] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269, 1995.

[37] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[38] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[39] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[40] Yu Guan and Thomas Plötz. Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(2):11, 2017.

[41] Nils Y Hammerla, Shane Halloran, and Thomas Plötz. Deep, convolutional, and recurrent models for human activity recognition using wearables. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1533–1540. AAAI Press, 2016.

[42] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[43] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360, June 2015.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European conference on computer vision*, pages 346–361. Springer, 2014.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[48] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[50] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[51] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017.

[52] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016.

[53] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

[54] Tâm Huynh and Bernt Schiele. Analyzing features for activity recognition. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, pages 159–163. ACM, 2005.

[55] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.

[56] Chadawan Ittichaichareon, Siwat Suksri, and Thaweesak Yingthawornsuk. Speech recognition using mfcc. In *International Conference on Computer Graphics, Simulation and Modeling (ICGSM'2012) July*, pages 28–29, 2012.

[57] J-SR Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3):665–685, 1993.

[58] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016.

[59] Michael Jones and Paul Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3(14):2, 2003.

[60] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5:954–975, 1993.

[61] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations*, 2018.

[62] Hak Gu Kim, Yeoreum Choi, and Yong Man Ro. Modality-bridge transfer learning for medical image classification. In *Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2017 10th International Congress on*, pages 1–5. IEEE, 2017.

[63] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[64] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4743–4751. Curran Associates Inc., 2016.

[65] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018.

[66] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[67] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[68] Mandar Kulkarni and Shirish Karande. Layer-wise training of deep networks using kernel similarity. *arXiv preprint arXiv:1703.07115*, 2017.

[69] Oscar D Lara, Labrador, and Miguel A. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys and Tutorials*, 15(3):1192–1209, 2013.

[70] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[71] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[72] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[73] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[74] Xiangang Li and Xihong Wu. Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4520–4524. IEEE, 2015.

[75] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[76] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[77] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[78] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[79] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[80] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Citeseer, 2013.

[81] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using convolutional auto-encoders with symmetric skip connections. *arXiv preprint arXiv:1606.08921*, 2016.

[82] Enrique S Marquez, Jonathon S Hare, and Mahesan Niranjan. Deep cascade learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[83] Daniel Martinho-Corbishley, Mark Nixon, and John N Carter. Super-fine attributes with crowd prototyping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[84] Daniel Martinho-Corbishley, Mark Nixon, and John N Carter. Super-fine attributes with crowd prototyping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[85] Zvika Marx, Michael T Rosenstein, Leslie Pack Kaelbling, and Thomas G Dietterich. Transfer learning with an ensemble of background tasks.

[86] Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *INTERSPEECH*, pages 3771–3775, 2013.

[87] Marvin Minsky and Seymour Papert. Perceptrons : an introduction to computational geometry, 1969.

[88] P Moeskops and JPW Pluim. Isointense infant brain MRI segmentation with a dilated convolutional neural network. In *MICCAI Grand Challenge on iSeg-2017: 6-month infant brain MRI Segmentation*, 2017.

[89] Fernando Moya Rueda, René Grzeszick, Gernot A Fink, Sascha Feldhorst, and Michael ten Hompel. Convolutional neural networks for human activity recognition using body-worn sensors. In *Informatics*, volume 5, page 26. Multidisciplinary Digital Publishing Institute, 2018.

[90] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[91] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 722–729. IEEE, 2008.

[92] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

[93] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International Conference on Machine Learning*, pages 2642–2651, 2017.

[94] Francisco Javier Ordonez, Gwenn Englebienne, Paula De Toledo, Tim Van Kasteren, Araceli Sanchis, and Ben Krose. In-home activity recognition: Bayesian inference for hidden markov models. *IEEE Pervasive Computing*, 13(3): 67–75, 2014.

[95] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1): 115, 2016.

[96] John Platt. A resource-allocating network for function interpolation. *Neural computation*, 3(2):213–225, 1991.

[97] Thomas Plötz, Nils Y Hammerla, and Patrick Olivier. Feature learning for activity recognition in ubiquitous computing. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1729, 2011.

[98] H Ponce, L Miralles-Pechuán, et al. A novel wearable sensor-based human activity recognition approach using artificial hydrocarbon networks. *Sensors (Basel, Switzerland)*, 16(7), 2016.

[99] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[100] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[101] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[102] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*, pages 108–109. IEEE, 2012.

[103] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[104] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, et al. Collecting complex activity datasets in highly rich networked sensor environments. In *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*, pages 233–240. IEEE, 2010.

[105] Daniel Roggen, Luis Ponce Cuspinera, Guilherme Pombo, Falah Ali, and Long-Van Nguyen-Dinh. Limited-memory warping lcss for real-time low-power pattern recognition in wireless nodes. In *European Conference on Wireless Sensor Networks*, pages 151–167. Springer, 2015.

[106] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[107] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[108] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[109] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al.

Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[110] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *CVPR 2011*, pages 1665–1672. IEEE, 2011.

[111] Tom Sercu and Vaibhava Goel. Dense prediction on sequences with time-dilated convolutions for speech recognition. *arXiv preprint arXiv:1611.09288*, 2016.

[112] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Joshua Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2107–2116, 2017.

[113] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[114] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[115] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4):323–339, 1992.

[116] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.

[117] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[118] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140. ACM, 2015.

[119] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[120] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[121] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[122] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[123] Matthew E. Taylor, Gregory Kuhlmann, and Peter Stone. Accelerating search with transferred heuristics. In *ICAPS-07 workshop on AI Planning and Learning*, September 2007.

[124] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[125] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[126] Di Wang, Edwin Candinegara, Junhui Hou, Ah-Hwee Tan, and Chunyan Miao. Robust human activity recognition using lesser number of wearable sensors. In *Security, Pattern Analysis, and Cybernetics (SPAC), 2017 International Conference on*, pages 290–295. IEEE, 2017.

[127] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1578–1585. IEEE, 2017.

[128] Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. ISBN 0-13-004029-0.

[129] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[130] Jelmer M Wolterink, Tim Leiner, Max A Viergever, and Ivana Išgum. Dilated convolutional neural networks for cardiovascular mr segmentation in congenital heart disease. In *Reconstruction, Segmentation, and Analysis of Medical Images*, pages 95–102. Springer, 2016.

[131] Shuochao Yao, Shaohan Hu, Yiran Zhao, Aston Zhang, and Tarek Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th International Conference on World Wide Web*, pages 351–360. International World Wide Web Conferences Steering Committee, 2017.

[132] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[133] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 636–644. IEEE, 2017.

[134] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[135] Yusen Zhan and Mattew E Taylor. Online transfer learning in reinforcement learning domains. In *2015 AAAI Fall Symposium Series*, 2015.

[136] Yan Zhang, Jonathon Hare, and Adam Prgel-Bennett. Learning to count objects in natural images for visual question answering. In *International Conference on Learning Representations*, 2018.

[137] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6230–6239. IEEE, 2017.

[138] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *International Conference on Web-Age Information Management*, pages 298–310. Springer, 2014.