On Feature Saliency and Deep Neural Networks

by

Yash Garg

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctorate in Philosophy

Approved November 2020 by the
Graduate Supervisory Committee:

K. Selçuk Candan, Chair
Hasan Davulcu
Baoxin Li
Maria Luisa Sapino

ARIZONA STATE UNIVERSITY

December 2020

ABSTRACT

Technological advances have allowed for the assimilation of a variety of data, driving a shift away from the use of simpler and constrained patterns to more complex and diverse patterns in retrieval and analysis of such data. This shift has inundated the conventional techniques and has stressed the need for intelligent mechanisms that can model the complex patterns in the data. Deep neural networks have shown some success at capturing complex patterns, including the so called attentioned networks, have significant shortcomings in distinguishing what is important in data from what is noise. This dissertation observes that the traditional neural networks primarily rely solely on the gradient-based learning to model deep features maps while ignoring the key insight in the data that can be leveraged as a complimentary information to help learn an accurate model. In particular, this dissertation shows that the localized multi-scale features (captured implicitly or explicitly) can be leveraged to help improve model performance as these features capture salient informative points in the data.

This dissertation focuses on "*working with the data, not just on data*", i.e. leveraging feature saliency through pre-training, in-training, and post-training analysis of the data. In particular, non-neural localized multi-scale feature extraction, in images and time series, are relatively cheap to obtain and can provide a rough overview of the patterns in the data. Furthermore, localized features coupled with deep features can help learn a high performing network. A *pre-training* analysis of sizes, complexities, and distribution of these localized features can help intelligently allocate a user-provided kernel budget in the network as a single-shot hyper-parameter search. Additionally, these localized features can be used as a secondary input modality to the network for cross-attention. Retraining pre-trained networks can be a costly process, yet, a *post-training* analysis of model inferences can allow

for learning the importance of individual network parameters to the model inferences thus facilitating a retraining-free network sparsification with minimal impact on the model performance. Furthermore, effective *in-training* analysis of the intermediate features in the network help learn the importance of individual intermediate features (neural attention) and this analysis can be achieved through simulating local-extrema detection or learning features simultaneously and understanding their co-occurrences. In summary, this dissertation argues and establishes that, if appropriately leveraged, localized features and their feature saliency can help learn high-accurate, yet cheaper, networks.

DEDICATION

“यह शोध प्रबंध मेरे माता–पिता और मेरी प्रिय बहन, रीमा, को समर्पित है।”

“*This dissertation is dedicated to my parents, and my dear sister, Reema.*”

ACKNOWLEDGEMENTS

I find myself short of words to be able to express my deep sense of gratitude and deference to my Ph.D. advisor, Dr. K. Selçuk Candan. I joined his research group in Fall 2013 as a Masters Thesis student, and since then he has acted on various roles to help me navigate graduate school journey. It was his unyielding support that acted as a guiding light during my Ph.D. journey. Achieving this milestone was only possible because of his invaluable feedback and critical insights in the topic. Dr. Candan, you have been and will be my constant source of inspiration for dedication, perfection and curiosity.

I want to give special thanks to Dr. Maria-Luisa Sapino for her countless and invaluable input on my work that has helped me make it complete and comprehensive. I would also like to thank Dr. Baoxin Li and Dr. Hasan Davulcu for being on my committee and giving their valuable guidance for shaping my dissertation.

I would like thank my parents for their unflagging love and for providing me with the means for a world class education. My dear sister, Reema, you are a great friend, your unconditional love and support has helped me build the path towards this accomplishment.

I would like give a massive shout out to my friends, Gaurav, Parag, Ashish, Luigi, Emre, Yaming (Lukas), Kevin, Hans, Mao-Lin, Jiayong, Nick and Sam for making my Ph.D. journey memorable and for always been there for me throughout this journey. I would also like to thank my EmitLab members, Parth, Jung, Xinsheng, Xilun, Shengyu, Sicong, Manjusha, Tasneema, Manoj, Magesh, and Shadab, and acknowledge that it was a pleasure working with you all.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Advancements in software and hardware technologies has enabled large-scale integration of cost-effective smart technologies that allow for the assimilation of a data in wide variety of applications, from images to video to time series. This explosion in the pace of availability of data has allowed for wide variety of multimedia analytical tasks, such as:

- **image classification**, supervised or unsupervised, involves learning a classifier that can map an image to a target label with a high probability, often referred as probabilistic classifier [34, 46, 52, 64, 101, 106, 113, 121, 132]
- **object detection** is one of the most highly researched problem in the present time with the advent of self-driving technologies. Object detection involves located object of interest an image, maybe even classify them [70, 102]
- **video analytics** involves learning a classifier that can work with a sequence of images (ordered by time) can track the movement of subjects across videos or classify gestures [57, 105]
- **recommendation system** aim at identifying personalized information given a prior knowledge on the user. Furthermore, they aim devising a novel ranking measure to rank the individual results w.r.t to the personal relevance [18]
- **language processing** has proven a great help in the rise of global communication in 21^{st} century. Language is no more barrier in how we communicate with our counter-parts across the world, for example, real-time language

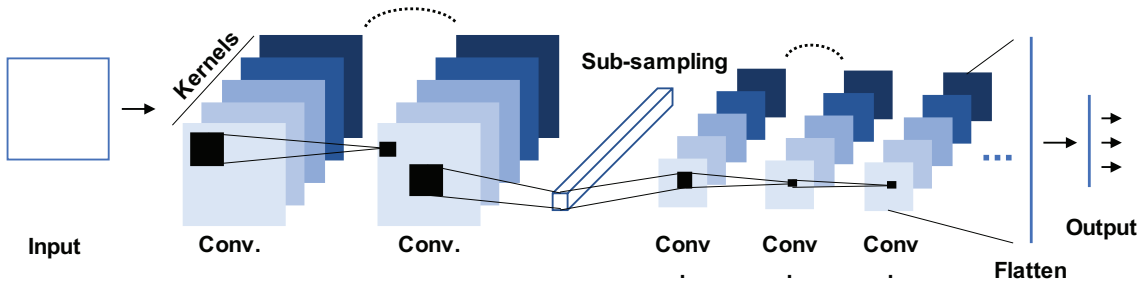


Figure 1.1: Overview of the Convolutional Neural Network Architecture

translation and image to text conversion [78, 85]

- **sentiment analysis** has been seen as a measure to understand the sentiments of people in online communication, particularly on social media. Sentimental analysis has allowed for emotion recognition and mental health monitoring through in-depth analysis of how people communicate on social media platforms [24, 100]

- **time series analytics** has taken a turn for prominence with the arrival of smart devices and wearable IoTs. This has opened the opportunities for heath monitoring, activity recognition and more more [24, 74, 81**? ?** , 126, 130]

Succeeding at any of these applications, particular in multimedia analytics, involves learning a function f that can maps the input $\boldsymbol{X}$ to target output $\boldsymbol{Y}$ using model parameters $\boldsymbol{W}$, s.t. $f : \boldsymbol{X}|\boldsymbol{W} \rightarrow \boldsymbol{Y}$. The ultimate goal here is to minimize the separation (loss), $\triangle(\boldsymbol{Y}, \boldsymbol{Y}^{'})$, between the observed $\boldsymbol{Y}$ and predicted $\boldsymbol{Y}^{'}$, i.e. $\boldsymbol{Y}^{'} = f(\boldsymbol{X}|\boldsymbol{W})$, here, $\boldsymbol{Y}$ can be a categorical observation (clustering, hinge loss [98]), or a probability distribution (classification, cross-entropy) or a real-valued observation (regression, mean absolute error).

1.1 Multimedia Retrieval

Multimedia retrieval tasks, such as the problem of image and time series classification have been one of the fundamental problems in the vision community that has attracted significant attention in the past *two* decades [25, 63, 66, 76]. Various works have been proposed through the use of global features [29], local features [9, 76], and, more recently, through deep features [106, 116] for learning effective and discriminating representation for learning highly accuracy classification models. Global features are aimed at learning the global representation (overall) of the input data, in contrast to the local features that aim at detecting salient feature points in the data (images or time series) that might be of interest. However, both of these approaches often lack the ability to generalize patterns contained in the data as a whole. Furthermore, these localized features (SIFT and UVTF, see details in Section 2) require a parametric bag-of-word approach to represent the individual instances of the data (an image or a time series). Deep features overcome this problem by their ability to learn a generalized representation in form of deep feature maps. Additionally, both global and local image features depend on an advanced classification model that feeds on these features to learn a classifier. One such example of image classification model is deep neural networks, $\mathcal{N}$, including convolutional and recurrent neural networks (CNNs) - see Figure 1.1 - have seen successful application in face recognition [64] as early as 1997, and in many more diverse applications in the recent years, such as time series analysis [130, 121], speech recognition [47], object recognition [70, 102, 101], and video classification [57, 105]. More importantly, CNNs' successful application in a variety of multimedia domain has led to a shift away from conventional feature-driven approaches such as local features (scale-invariant feature transform - SIFT [76] and Speed-Up Ro-

bust Features - SURF [9]) and global features (histogram-of-gradients - HOG [25]) towards learning deep generalized features using a well-crafted high performing deep network architectures.

1.1.1 Deep Neural Networks as Classifiers

Deep networks have unequivocally shown their high performance by intelligently learning a variety of deep feature maps that capture a variety of salient information from the input data, such as pattern shapes (highly abstract) to pattern textures (moderately abstract) to pattern colors (finely detailed). Deep networks, in general, are a class of supervised learning models that rely on gradient-based learning for capturing the abstract features in the data that can help the model classifier the input to a particular target with confidence, also known as probabilistic classification.

The design, architecture, and the type of computation with the network, of deep networks involves a significant amount of expert knowledge and hand-tuning, it is search for optimal hyper-parameter configuration or design of neuron (the fundamental computing units in the network) remains *an art form, than science*. Various works have been done for hyper-parameter search, such as grid search [63] and random search [10], or by devising novel components for the networks such as, ReLU [80], and batch-normalization [55]. However, the search for high-performing network configuration remains an expert-driven art-form rather than a science, as more recently, the search for hyper-configuration has moved towards searching more specialized architectures, such as VGG [106], DenseNet [52], and RESNet [46].

Given the success of specialized architectures, as seen earlier, contemporary deep networks owe their success to the *depth* and *width* of the networks - thanks to a large number of trainable parameters - that helps learn complex patterns con-

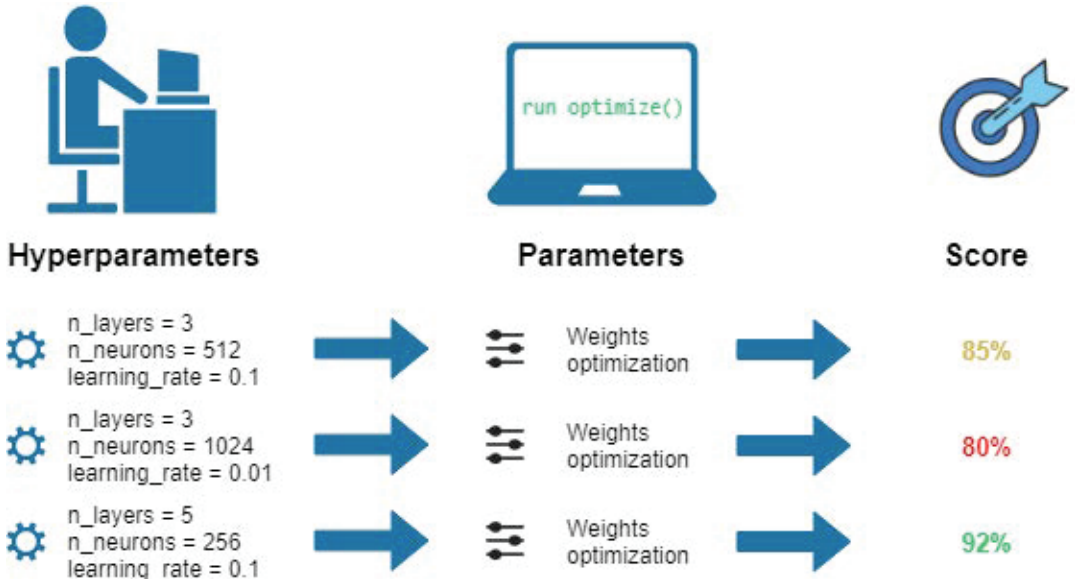


Figure 1.2: A Typical Flow of Hyper-Parameter Search Process [60]

tained in the image classification datasets, such as ImageNet [62]. Often, the number of hyper-parameters in these specialized network architectures range from tens of thousands [66] to hundred of million [106, 52, 46], and such a large parameter space may lead to a significant amount of parameters that are redundant and irrelevant to the final network output (performance). These redundant and irrelevant hyper-parameters can have an insignificant or adverse contribution to the network by introducing noisy information in the overall networks.

Much of the focus while working with deep neural networks have always been focused on searching and pruning of the network parameters, inadvertently the hyper-parameters, while assuming that both the input and latent features in the networks have equal importance and contribute equally to the final network output. Recently, neural networks have borrowed the concept of the feature attention from the natural language processing that highlights that, "*not all input features contribute equally to the output features*" but there exists a contextual relationship between the input and the output features [7]. Since 2014, after Bahdanau [7], significant

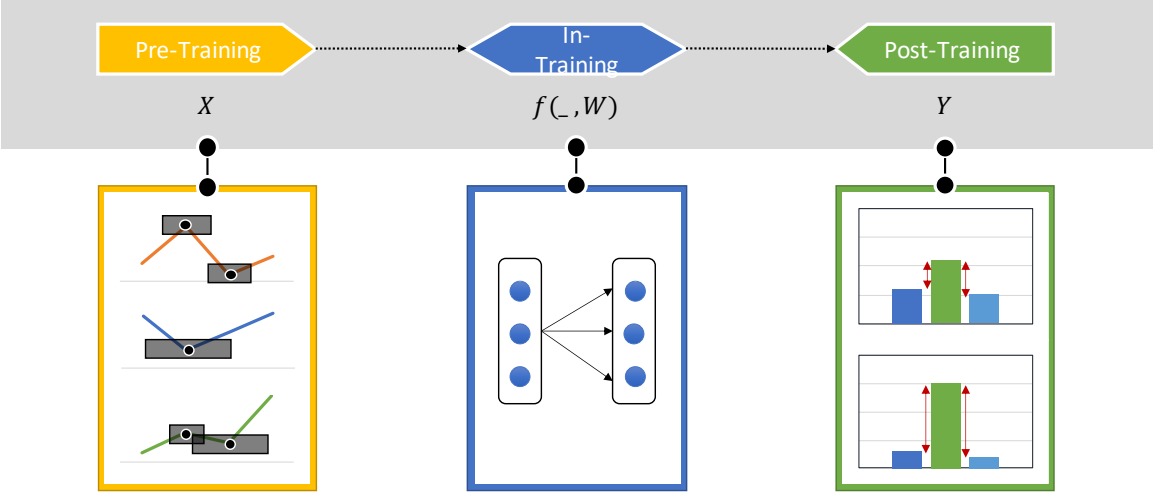


Figure 1.3: Overview of the Various Data Analysis Opportunities While Training a Deep Neural Network to Learn a Highly-Accurate Model

amount of work has been carried out for learning intelligent attention mechanisms, such as Self Attention [79], Squeeze-and-Excitation Network (SENet) [51], Convolutional Block Attention Module (CBAM) [119], Bottleneck Attention Module (BAM) [91], Fusion Attention Networks (FusAtNet) [84], Transformer [114], and Multi-Scale Multi-Head Attention (MSMSA) [40], and have shown that the leveraging variety of relationship in the deep features, such as spatial (images), temporal (time series) and filter (neurons), can help inform an effective attention mask that can boost the network performance. However, attention modules have often relied on the network's ability to learn a generalized represents of the data and in-turn update the attention mask during the back-propagation phase.

This dissertation observes that instead of solely relying on expert inputs and hand-tuning of the networks (be it hyper-parameters or connections in the network) to learn a highly-accurate model, one can instead focus on understanding and analyzing the data available before, while, and after training the model, i.e. pre-, in- and post-training analysis of the data.

1.2 Data Analysis for Learn Accurate Networks

As noted in the previous section, there is a vast amount of information that is available while training a neural network. While one initially starts with input data and the associated target data, during training the network generates hidden data and intermediate network parameters can be further offer insights on the network's capability and consequently used to improve the model performance. Furthermore, the model output, i.e. model inferences, is often considered only suitable for measuring model loss, but can in-fact offer more than loss. An intelligent analysis can help understand the classification power (if not accuracy) of the models. Therefore, as shown in Figure 1.3, there are *three* critical opportunities to analyze data that can offer insights to help with model training.

1. **Pre-Training Analysis** is used to gather insights (salient information) from the inputs and leverage the gathered insights in the model
2. **In-Training Analysis** is used to understand the how internal model activations (hidden data) are working and how they can be improved by novel operations that can help improve their overall impact on the model performance
3. **Post-Training Analysis** is used to measure the classification power of the model by understanding the model inferences

Neural Networks, in particular, are type of *gradient-based learning algorithm* [66] that aim to learn a mapping from input data ($\boldsymbol{X}$) to the target output ($\boldsymbol{Y}$) using trainable model parameters ($\boldsymbol{W}$). As [10, 36] describes, the ultimate goals of a neural network is to learn a function (f) s.t.

$$f : \boldsymbol{XW} \rightarrow \boldsymbol{Y} \tag{1.1}$$

in other words, the above mapping can be written as,

$$\boldsymbol{Y} \approx f(\boldsymbol{X}, \boldsymbol{W}) \tag{1.2}$$

Therefore, *pre-training* analysis can be seen as drawing deductions from the input data ($\boldsymbol{X}$), *in-training* analysis can be seen as understanding the intermediate operations ($f(\_, \boldsymbol{W})$), and *post-training* analysis can be seen as understand the model output/inferences ($f(\boldsymbol{X}, \boldsymbol{W})$).

1.2.1 Pre-Training Analysis

The key vulnerability of neural network remains in their need for a new hyper-parameter configuration for every unique dataset and the presence of noise, which is a very common occurrence in the real-world datasets. This dissertations, we the works presented in the Chapter 3 and **??**, argues that if data is analyzed intelligently before the model is trained, one can extract key insights that vary in size, complexity, distribution, and diversity to either allocated a user provide kernel budget for single-shot hyper-parameter configuration search or learn a complimentary additional input modality that can help reduce the noise in the original input can be used a cross-attention mechanism.

In particular, a principled approach for understanding the sizes, complexities, distribution, and diversity of the complex local patterns contained in the input data can uncover information that might otherwise have been overlooked. This principled approach is relatively cheaper to perform in comparison to training hundreds of hyper-parameter configurations. Furthermore, these localized patterns can help separate relevant information from irrelevant (noisy) information for the network to focus on (attention). These two strategies can help learn a highly accurate neural network with minimal computational overhead.

1.2.2 Post-Training Analysis

The increase in *depth* and *width* of the neural networks have been one of the primary factors for the success of deep networks, however, this increase has led to an exponential increase in the number of trainable parameters in the network. Consequently, leading to a significant increase in the need for computational resources and has introduced redundant and irrelevant parameters. Works such as Dropout [108], PFEC [69] and NISP [123] have shown that the network contains significant amount of redundancy in the network. Dropout [108] introduced the concept of randomly, but temporally, dropping the neuron activations during the feed-forward phase of network training to prevent network overfitting by reducing the co-variance of the neuron i.e. reducing the neuron co-dependence. Furthermore, these strategies classify the neurons into three categories:

- **Significant Neurons**: positively impact the model output
- **Neutral Neurons**: neither positively nor negatively impact the model output
- **Insignificant Neurons**: negatively impacts the model output

Here, aim is to penalize the insignificant and neutral neurons more than the significant ones to push the formers to significance category. Furthermore, these strategies do not account for the impact of removing either neuron or the weights (parameters) to overall network output. Additionally, these methods can only be used to. Therefore, a post-training analysis can leverage the model inference to deduce the impact of model parameters on a pre-trained model and identify candidate model parameters that can be sparsified without having to re-train the network, i.e. saving significant amount of computational expense.

1.2.3 In-Training Analysis

With the increasing size of the neural networks to complement the complexity and size of modern datasets, the neural networks are being inundated by sheer size in hidden activations, consequently experience a deterioration in model accuracy. [7] observed that one can model the relationship in intermediate network activation and learn regions (local area) to focus on, i.e. attention. Mathematically, attention is a way of weighting the input features by their learned importance as follows: $\boldsymbol{Y} = \sigma(\boldsymbol{W}\boldsymbol{X} + b) \odot \boldsymbol{M}_a$, where, $\boldsymbol{M}_a$ is the attention mask s.t. $\boldsymbol{M}_a \epsilon [0, 1]$ (soft attention) or $\boldsymbol{M}_a \epsilon 0, 1$ (hard attention). To learn the mask matrix ($\boldsymbol{M}_a$), a neural network is modified to have two branches, (1) feature branch and (2) mask branch. The *feature branch* is the conventional feed-forward branch that learns the latent features to map input ($\boldsymbol{X}$) to the output ($\boldsymbol{Y}$), whereas, the *mask branch* aims to learn the contextual relationship between the input and output features - represented as $\boldsymbol{M}_a$, the attention scores.

However, conventional does not account for the change in the latent features in two adjacent layers, but rather attention module for each layer is mutually exclusive with other modules in the network. It can be argued that the change in latent features across layers highlights the local regions in the images that are important to understand the patterns contained in the images, therefore, the mask branch can be adapted to take latent features from two layers to learn informed attention. Furthermore, learning attention explicitly as a particular layer limits the multi-scale information being account for, as complex patterns are often composed of multiple simpler patterns. Highlighting the need for multi-scale attention masks that account for a multitude of scales of patterns while learning attention masks.

1.3 Dissertation Outline

The remainder of the dissertation is organized as follows:

- In Chapter 2 presents the overview of various type of features, localized, latent and deep, considered in this dissertation
- In Chapter 3, presents RACKNet a novel pre-training analysis of localized image features to allocate user provided kernel budget
- In Chapter 4 presents iSparse framework, a post-training analysis of model inferences (model outputs) for re-training free network sparsification
- In Chapter 5 presents the scale-space based attention network for local feature driven attention mechanism
- In Chapter 6 presents the novel SDMA mechanism to learn a complimentary saliency series for cross attention in time series
- In Chapter 7 present XM2A, a unique multi-scale multi-head attention with cross-talk framework, to learn temporal attention by leveraging the co-existence of local multi-scale features
- Chapter 8 concludes the dissertation.

Chapter 2

BACKGROUND

As seen in Chapter 1, successful applications of the deep networks in variety of domains has lead to a significant shift away from the localized feature based approaches, such as Scale Invariant Feature Transform (SIFT) [76] and Uni-Variate Temporal Features (UVTF) [15]. While deep representation owe their success to learning rich and generalized representation of the patterns contained in the data through the use to large and complex network architecture, such as, VGG [106], ResNet [46] and InceptionNet [110], they still remain susceptible to noise and are expensive to obtain, where as localized features are cheaper to obtain and can provide rough overview of the data that can help learn a high performing network [32]. Furthermore, latent features have shown promising results in identifying latent basis vector (linearly independent eigenvectors) that maximized the separation between the data points. This chapter discuss the literation on localized features ()and latent features that fundamental to this dissertation.

2.1 Localized Features

2.1.1 Scale-Invariant Feature Transform (SIFT)

The Scale-Invariant Feature Transform (SIFT) [76] was devised to identify salient keypoints in the images and describe them in a specialized representation, *feature descriptors*, that are robust to scaling, translation, and rotation of pattern and image brightness. These characteristics of SIFT has made it the *de-facto* local image feature extraction strategy for content-based image retrieval.

SIFT[1] relies on a multi-step process to extract stable and scale-invariant patterns contained in a given image, intuitively, SIFT keypoints corresponds to regions in a given image that are different from their neighborhood, the steps are as follows:

1. **Scale-Space Generation** – Image data is transformed into multi-scale of different sizes determined by the Gaussian kernel used. This allows for identification of the features are different scales (see Section 2.1.1.1)

2. **Extrema Detection and Localization** – In this step, a search is carried out in all image scales and spatial locations in the images over the difference-of-Gaussian, generate in the prior step, to detect point of interests. Once the interest points are detected, they localized at a scale and spatial location where they are most stable (see Section 2.1.1.2)

3. **Orientation Assignment** – Prominent gradient direction is determine at the location and scale of point-of-interest (see Section 2.1.1.3)

4. **Descriptor Generation** – Local gradients around the point of interest are measures and summarized as histogram-of-orientations, gradients represents the direction of the change in neighboring points. this allows for capturing the local shape, distortion and illumination to provide rotation, scale and illumination invariance (see Section 2.1.1.4)

Given these steps, is named "*Scale Invariance Feature Transform* (SIFT)" as the point-of-interest is transformed into a feature representation that is robust to change in scale, orientation, and illumination.

[1]More details can be found in "Lowe, D. G., "Distinctive image features from scale-invariant keypoints", International Journal of Computer Vision 60, 2, 91–110 (2004)."

2.1.1.1 Scale-Space Generation

The key aspect of the SIFT keypoints[2], scale-invariant, can be attributed to the multi-scale representation generated using the principled approach proposed by Witkin [118]. Lowe in SIFT, extended the Witkin's proposed "*principled approach to describe a signal qualitatively, managing the ambiguity of scale in an organized and natural way*" for images [118].

Specifically, Lowe in [76], proposed to expand the input image, along scale (σ), by convolution with Gaussian masks (G) over a continuum of sizes, i.e. scale . The choice of using Gaussian mask is driven by various factors:

- **monotonic**: symmetric around mean, and strictly decreasing at distance from mean increases
- **well-behaved**: for each observed value for scale (size) parameter has a non-zero probability density
- **locally sufficient**: same probability distribution can be apply to both true sample and the sample observed at a given scale

Thus Gaussian masks satisfies the "*well-behavedness*" criteria, i.e. it is monotonic, well-behaved, and locally sufficient.

Gaussian kernel for a given image, $\boldsymbol{I}$, depends on *three* factors, the independent image variables, w and h (the width and height pixel coordinate), and scale, σ, Gaussian's standard deviation, is as follows:

$$G(w, h, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(w^2+h^2)/2\sigma^2}, \tag{2.1}$$

[2]For the purposes of this dissertation, SIFT features would be referred to a "*keypoints*" to avoid ambiguity with deep features and dimensionality (features).

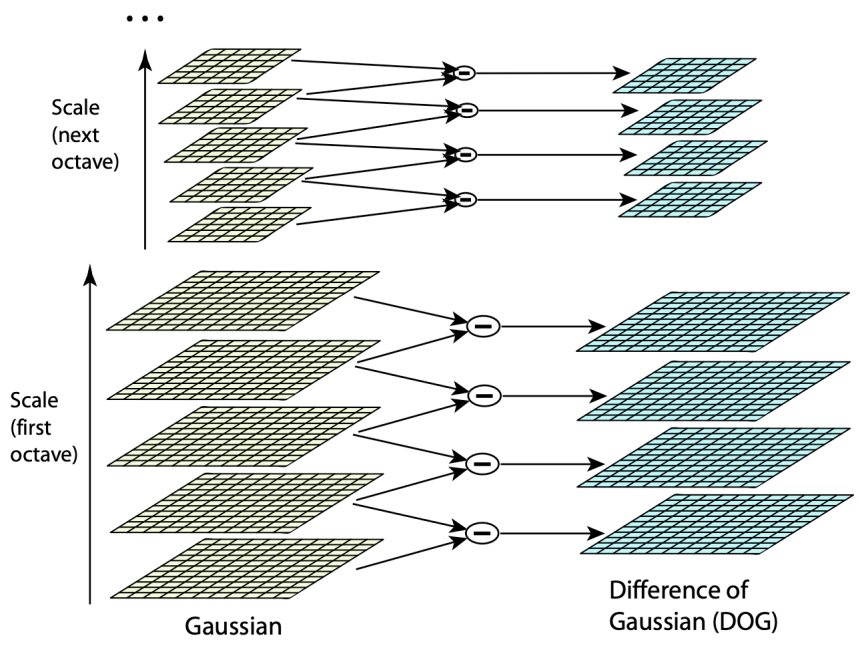


Figure 2.1: Scale-Space Pyramid Generation by Smoothing an Image with Gaussian Masks of Different Sizes. The Color"*Yellow*" (left) Represents the Gaussian Generated and the Color "*Blue*" (right) Represents the Difference-of-Gaussian [76]

and the Gaussian convolved image representation, $\boldsymbol{L}$, can be written as:

$$\boldsymbol{L}(w, h, \sigma) = G(w, h, \sigma) \circledast \boldsymbol{I}(w, h), \tag{2.2}$$

here, $\circledast$ represents the convolution operation with respect to w and h. As Witkin noted, "this function defined a hyper-surface on the (w, h, σ)-hyper-plane, where each profile of constant σ is a Gaussian-smoothed version of $\boldsymbol{I}(w, h)$, the amount of smoothing proportional to σ." Thus referred to as Gaussian Scale-Space (GSS), scale from σ and space from w and h.

Next, Lowe argues that stable keypoint can be identified in the scale-space using the difference-of-Gaussian (DOG), as difference-of-Gaussian is a close approximation of scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$ leading to most stable extrema (minima or maxima) being detected. Therefore, difference-of-Gaussian

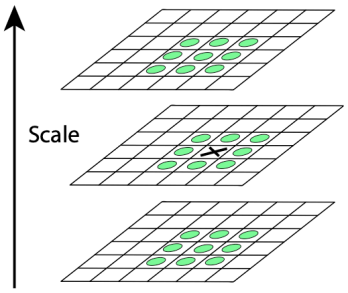


Figure 2.2: Extrema Detection Long Space and Scale. The Symbol $\times$ Represents the Point of Interest in the Images, and the "*Green*" Circles are the Neighbors [76]

(DOG), $\boldsymbol{D}$,can be defined as difference of two adjacent scaled separated by multiplicative factor k as follows:

$$\begin{aligned}\boldsymbol{D}(w,h,\sigma) &= (G(w,h,k\sigma) - G(w,h,\sigma)) \circledast \boldsymbol{I}(w,h) \\ &= \boldsymbol{L}(w,h,k\sigma) - \boldsymbol{L}(w,h,\sigma)\end{aligned} \tag{2.3}$$

Furthermore, the uses of difference-of-Gaussian, helps eliminate high-frequency details, such as random noise, and highlight the edges appearing at various scales. Thus, Difference-of-Gaussian (DOG) can be seems as an enhancement strategy that involves the subtraction of one blurred image ($G(w,h,k\sigma)$) representation from another blurred version ($G(w,h,\sigma)$).

2.1.1.2 Extrema Detection and Localization

With the successful generation of the Gaussian Scale-Space (GSS) (see Figure 2.1), next step is to detect the local extrema keypoints, i.e. local minima and local maxima. This detections step is performed in the $\boldsymbol{D}(k^s\sigma)$, by comparing each local point, $\langle w,h\rangle$, to their *eight* spatial neighbors on *eighteen* scale neighbors (*nine* in each predecessor ($\boldsymbol{D}^{k^{s-1}}$) and successor scale ($\boldsymbol{D}^{k^{s+1}}$)), as shown in Figure 2.2

and formalized in Equation 2.4.

$$max \left(\begin{array}{ccc} \boldsymbol{D}^{w-1,h-1,k^{s+1}\sigma} & \boldsymbol{D}^{w,h-1,k^{s+1}\sigma} & \boldsymbol{D}^{w+1,h-1,k^{s+1}\sigma} \\ \boldsymbol{D}^{w-1,h,k^{s+1}\sigma} & \boldsymbol{D}^{w,h,k^{s+1}\sigma} & \boldsymbol{D}^{w+1,h,k^{s+1}\sigma} \\ \boldsymbol{D}^{w-1,h+1,k^{s+1}\sigma} & \boldsymbol{D}^{w,h+1,k^{s+1}\sigma} & \boldsymbol{D}^{w+1,h+1,k^{s+1}\sigma} \\ \hline \boldsymbol{D}^{w-1,h-1,k^{s}\sigma} & \boldsymbol{D}^{w,h-1,k^{s}\sigma} & \boldsymbol{D}^{w+1,h-1,k^{s}\sigma} \\ \boldsymbol{D}^{w-1,h,k^{s}\sigma} & \boldsymbol{D}^{w,h,k^{s}\sigma} & \boldsymbol{D}^{w+1,h,k^{s}\sigma} \\ \boldsymbol{D}^{w-1,h+1,k^{s}\sigma} & \boldsymbol{D}^{w,h+1,k^{s}\sigma} & \boldsymbol{D}^{w+1,h+1,k^{s}\sigma} \\ \hline \boldsymbol{D}^{w-1,h-1,k^{s-1}\sigma} & \boldsymbol{D}^{w,h-1,k^{s-1}\sigma} & \boldsymbol{D}^{w+1,h-1,k^{s-1}\sigma} \\ \boldsymbol{D}^{w-1,h,k^{s-1}\sigma} & \boldsymbol{D}^{w,h,k^{s-1}\sigma} & \boldsymbol{D}^{w+1,h,k^{s-1}\sigma} \\ \boldsymbol{D}^{w-1,h+1,k^{s-1}\sigma} & \boldsymbol{D}^{w,h+1,k^{s-1}\sigma} & \boldsymbol{D}^{w+1,h+1,k^{s-1}\sigma} \end{array} \right) \tag{2.4}$$

Thus, pruning the local points that are similar to their local neighborhood, both in scale and space. In other words, a local point, $\langle w, h \rangle$, is a keypoint, $\mathcal{K}$, if it is greater than Θ% of the maximum of it's 26 scale-space neighbors in DOG.

Next step, is to identify if the detection keypoints have low contrast (sensitive to noise) or are poorly localized along as edge. this is achieved by the use of Taylor series expansion of the scale-space function, $\boldsymbol{D}(w, h, \sigma)$, s.t.

$$\boldsymbol{D}(x) = \boldsymbol{D} + \frac{\partial \boldsymbol{D}^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 \boldsymbol{D}^T}{\partial x^2} x. \tag{2.5}$$

Here, $x = \langle w, h, \sigma \rangle$. The localized position, $\hat{x}$, of the keypoint is determined by taking the derivative of the function with w.r.t x,

$$\hat{x} = -\frac{\partial^2 D}{\partial x^2}^{-1} . \frac{\partial \boldsymbol{D}}{\partial x}, \tag{2.6}$$

thus, the $\boldsymbol{D}(\hat{x})$ can be computed as

$$\boldsymbol{D}(\hat{x}) = \boldsymbol{D} + \frac{1}{2} \frac{\partial \boldsymbol{D}^T}{\partial x} \hat{x}. \tag{2.7}$$

Lowe discards all extrema points where $\lfloor \boldsymbol{D}(\hat{x}) \rfloor < 0.03$, thus eliminating the points with low contrast, adding invariance to noise (contrast).

DOG leads to strong edge responses, thus making pruning solely based on low contrast insufficient. Therefore, poorly defined peaks in DOG can contain large principal curvature ratio, i.e. bending of the surface at the peak. The curvature can be computed using Hessian matrix, $\boldsymbol{H}$ at keypoint location and scale as follows:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{D}_{ww} & \boldsymbol{D}_{wh} \\ \boldsymbol{D}_{hw} & \boldsymbol{D}_{hh} \end{bmatrix} \tag{2.8}$$

The derivatives can be estimated by taking the difference of the neighboring sample points. The eigenvalues of $\boldsymbol{H}$ are proportional to the principal curvature of $\boldsymbol{D}$, relying on the Harris corner detector. Therefore, we can say that,

$$Tr(\boldsymbol{H}) = \boldsymbol{D}_{ww} + \boldsymbol{D}_{hh} = \alpha + \beta$$

$$Det(\boldsymbol{H}) = \boldsymbol{D}_{ww}\boldsymbol{D}_{hh} - (\boldsymbol{D}_{wh})^2 = \alpha\beta$$

Here, α and β represents the top *two* eigenvalues. If the determinant is negative, there the keypoint is rejected. Next,l et us assume, curvature ratio, r , the ratio of the *two* eigenvalues, s.t. $\alpha = r\beta$, then:

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \tag{2.9}$$

Next, goal is to pruning keypoint that have a curvature ratio higher than $\frac{(r+1)^2}{r}$, therefore,

$$\frac{Tr(\boldsymbol{H})^2}{Det(\boldsymbol{H})} < \frac{(r+1)^2}{r}. \tag{2.10}$$

Lowe uses $r = 10$ as the default values of r to pruning keypoint by curvature ratio.

2.1.1.3 Orientation Assignment

In this step, a principled approach is used to assign a consistent orientation, i.e. gradient direction, to each keypoint given their local neighbors. To determine the

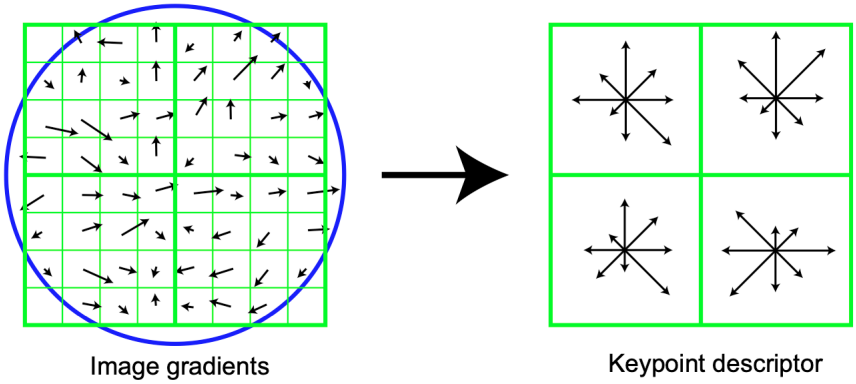


Figure 2.3: Keypoint Descriptor Extraction. "*Green*" Boxes Represents the Grid Cell Describing the Neighborhood Around the Point-of-Interest, "*Blue*" Circle Representing the Radius of the Scope of Keypoint, and "*Black*" Arrows Represents the Direction of the Gradients and Length of the Arrow Represents the Magnitude of the Gradient Change [76]

keypoint orientation, the scale associated with the keypoint is leveraged to determine the magnitude (m) and orientation (θ) of the gradient at keypoint of location:

$$m(w,h) = \sqrt{(\boldsymbol{L}(w+1,h) - \boldsymbol{L}(w-1,h))^2 + (\boldsymbol{L}(w,h+1) - \boldsymbol{L}(w,h-1))^2} \quad (2.11)$$

$$\theta(w,h) = tan^{-1}((\boldsymbol{L}(w+1,h) - \boldsymbol{L}(w-1,h))/(\boldsymbol{L}(w,h+1) - \boldsymbol{L}(w,h-1))) \quad (2.12)$$

2.1.1.4 Descriptor Generation

Descriptor can be seen as a vectorized representation of the local neighborhood gradient around the keypoint (see Figure). This neighborhood is defined by a 16×16 pixel neighborhood around the keypoint local $\langle w, h \rangle$. This neighborhood is than split into a 4×4 grid, where each grid cell is of size 4×4 pixels. An 8-bin histogram-of-orientation is created for each grid cell. Here, each bin in the histogram captures 45° s.t. all 8 bins together covers entire 360° angle spectrum, i.e. gradient orienta-

tion. Once the histogram is generated, it is can transformed into normal vector, i.e. descriptor to have unit magnitude.

2.1.2 Uni-Variate Temporal Features (UVTF)

[15] extended the SIFT for Uni-Variate Temporal Features (UVTF)[3], and showed localized temporal keypoint (extracted similar to SIFT) can be used to speed up expensive time series operations, such as DTW computation.

This section describes in details the localized keypoints extraction process (consisting of "scale-space generation" and "extrema detection" steps), motivated from SIFT, to identify key intervals (or "keypoints") in the individual variates in a multivariate time series.

2.1.2.1 Temporal Scale-Space Generation

Intuitively, the smoothing process can be seen as generating a multi-scale representation of the given series and thus the differences between smoothed versions of a given series correspond to differences between the same series at different scales. Based on the argument that the interesting events will be maximally different from the overall pattern in their local neighborhoods, searches for those points that have largest variations with respect to both time and scale. Therefore, the first step of the process is to create a scale-space consisting of multiple smoothed versions of a given series – each resulting series are then subtracted from the series in the adjacent temporal scale to obtain the what are referred to as difference-of-Gaussian series.

Let $\boldsymbol{T}_v$ represent a uni-variate time series, s.t. $\boldsymbol{T}_v \in \mathbb{T}[v,:]$, and $\boldsymbol{T}_v^{(t,\sigma)}$ represents

[3]More details can be found in "Candan, K. Selçuk, et al. "sDTW: Computing DTW Distances using Locally Relevant Constraints based on Salient Feature Alignments." VLDB (2012)."

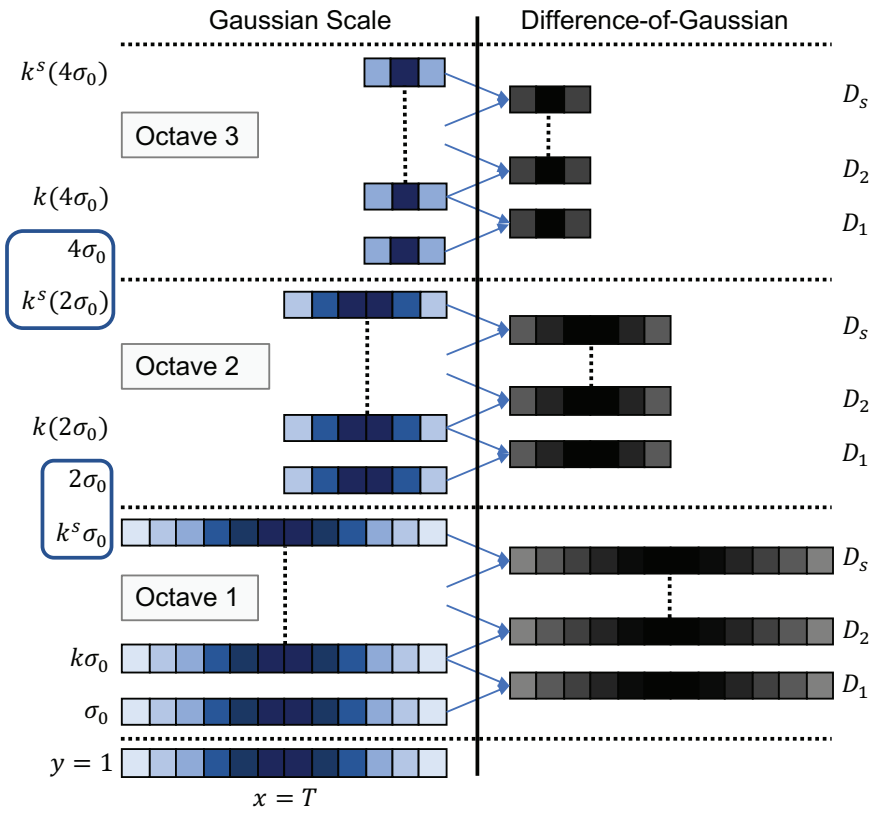


Figure 2.4: Generating Gaussian Scale-space and Dog for a Variate from a Multivariate Time Series. Here, y and x Represents the Variate and Temporal Dimensions of a Time Series

the smoothed version of $\boldsymbol{T}_v$ through convolution with the Gaussian function along the temporal dimension:

$$G\left(\mathrm{t}, \sigma\right) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-t^2}{2\sigma^2}} \tag{2.13}$$

such that

$$\boldsymbol{L}_v(t, \sigma) = G\left(\mathrm{t}, \sigma\right) \circledast \boldsymbol{T}_v(t). \tag{2.14}$$

Intuitively, Gaussian smoothing can be perceived as a multi-scale representation of a given series ($\boldsymbol{T}_v$), and the subsequent differences of the different Gaussian smoothed (Difference-of-Gaussian - DoG) series correspond to difference of the

Figure 2.5: A Sample Candidate Keypoints Point, $\mathcal{F}$, (Solid *Black*) and It's Neighbors in Adjacent Scales "$s+1$" (*Red*) and "$s-1$" (*Yellow*) and in Time "$t-1$" (*Blue*) and "$t+1$" (*Green*)

same series at difference scales. Therefore, the DoG can be computed as

$$\begin{aligned} \boldsymbol{D}_v(t,\sigma) &= (G(t,k\sigma) - G(t,\sigma)) \circledast \boldsymbol{T}_v(w,h) \\ &= \boldsymbol{L}_v(t,k\sigma) - \boldsymbol{L}_v(t,\sigma) \end{aligned} \tag{2.15}$$

Here, $\boldsymbol{D}_v$(t, σ is the difference between the representation of the same input series ($\boldsymbol{T}_v$) smoothed at different scales, σ and $k\sigma$ (here, k is the constant multiplicative factor). Analogous to SIFT, UVTF incrementally reduce the temporal length of the input series ($\boldsymbol{T}_v$), s.t.

$$\boldsymbol{T}_v(t,2\sigma) = \frac{\boldsymbol{T}_v(2*t-1,k^s\sigma + \boldsymbol{T}_v(2*t,k^2\sigma)}{2} \tag{2.16}$$

organized as octaves, $\mathcal{O}$, where each of these $\mathcal{O}_o,\ o \in \{1,...,O\}$ octaves is further organized as scales, $\mathcal{S}$ s.t. a given scale $\mathcal{S}_s,\ s \in 1,...,S$ has an associated smoothing factor of $k^s\sigma$ s.t. $k^S = 2$, within each octave o.

2.1.2.2 Extrema Detection

This step searches for points of interest, $\langle t,o,s\rangle$ across multiple scales of the given time series, v, by searching over multiple scales and locations of the given series (here o denotes an octave and s denotes the corresponding scale). The search of local extrema (keypoints) is performed by comparing the immediate neighbor (see

Figure 2.5) along both time and scale in the DoG representation, $\boldsymbol{D}_v^{(\mathrm{t},\sigma)}$, of the input series, $\boldsymbol{T}_v$. Thus, pruning the keypoints that are similar to their local neighborhood, both in scale and time. A keypoint, $\mathcal{K}\langle v,t,o,s\rangle$, is an extremum if it is maximum across it's *eight* neighbors, *three* is each adjacent scales ($s-1$ and $s+1$), and *two* in time ($t-1$ and $t+1$)) i.e.

$$max\left(\begin{array}{ccc} \boldsymbol{D}_v^{t-1,k^{s+1}\sigma} & \boldsymbol{D}_v^{t,k^{s+1}\sigma} & \boldsymbol{D}_v^{t+1,k^{s+1}\sigma} \\ \hline \boldsymbol{D}_v^{t-1,k^{s}\sigma} & \boldsymbol{D}_v^{t,k^{s}\sigma} & \boldsymbol{D}_v^{t+1,k^{s}\sigma} \\ \hline \boldsymbol{D}_v^{t-1,k^{s-1}\sigma} & \boldsymbol{D}_v^{t,k^{s-1}\sigma} & \boldsymbol{D}_v^{t+1,k^{s-1}\sigma} \end{array}\right) \tag{2.17}$$

In other words, $\langle t,o,s\rangle$ is designated as an extremum if it is greater than Θ% of the maximum of it's *8* scale-time neighbors in DOG ($\boldsymbol{D}$). Furthermore, each identified keypoint has an associated temporal keypoint scope, defined by the temporal scale (s) in which it is located. The radius of the keypoint is set to 3σ, as the under Gaussian smoothing *three* standard deviation would cover ~ 99.73% of the original temporal points that has contributed to the keypoint.

Given these, each key temporal keypoint can be rewritten as quadruple, $\mathcal{F}\langle v,t,o,s\rangle$.

2.2 Latent Features

Latent, word of latin origin "$lat\tilde{e}ns$" [**leyt**-nt], referred to as hidden. Latent features are variables or representations that are not observable directly, but inferred mathematically from other directly observed features/variables. Latent variables are often referred to as hidden categories - hypothetical variables/construct. This interpretation of latent variable has leads to their widespread use in the domain of dimensionality reduction, through approaches such as Principal Component Analysis (PCA) [92] and Infinite Feature Selection (InfFS) [97], aimed at learning orthonormal latent features that maximizes the variance between the data points and separation between the features respectively.

2.2.1 *Principle Component Analysis (PCA)*

Principal Component Analysis (PCA) is eigen-decomposition based latent features extraction approach and can be seen as an approach that fits d-dimensional latent space over a D-dimensional original space.

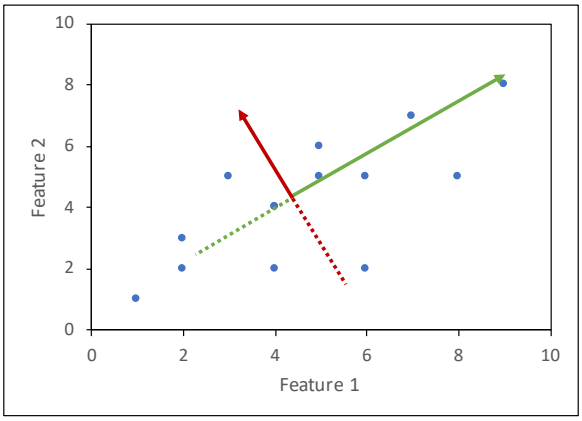


Figure 2.6: Abstract Overview of PCA Applied on 2D Data Points, and Shows Two Prominent Orthonormal Principal Components."*Green*" and "*Red*" Represents the 1^{st} and 2^{nd} Principal Components Respectively

PCA can be defined as orthonormal linear transformation that transformed the original features space into new latent space such that the greatest variance by some scalar projection of the data comes to lie on the first latent dimensional, also called "*principal component*".

Given a data matrix, $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, where each feature (matrix column) is centered around zero. Mathematically, the input matrix is first converted into a covariance matrix, $\boldsymbol{X}^c$, s.t.

$$\boldsymbol{X}^c = \boldsymbol{X}\boldsymbol{X}^T, \tag{2.18}$$

and the $\boldsymbol{X}^T$ can be transformed into

$$\boldsymbol{X}^T = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}, \tag{2.19}$$

here, $\boldsymbol{U}$ and $\boldsymbol{V}$ are left and right eigenvectors and $\boldsymbol{S}$ is the Eigenvalue matrix. The input data covariance matrix, $\boldsymbol{X}^T \in \mathbb{R}^{D \times D}$ is converted into *three* matrix as follow: $\boldsymbol{U} \in \mathbb{R}^{D \times d}$,$\boldsymbol{S} \in \mathbb{R}^{d \times d}$, and $\boldsymbol{V} \in \mathbb{R}^{d \times D}$, where d represents the number of reduced dimensions or latent features. Mathematically, $\boldsymbol{U} = \boldsymbol{V}^T$ as both dimensions of the $\boldsymbol{X}^T$ represents features.

Another extension of PCA is the Singular Vector Decomposition (SVD), this approach works on the data matrix directly as opposed to covariance matrix. Analogous to PCA, SVD is defined as follows:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}, \tag{2.20}$$

here, the input matrix, $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ is converted into *three* matrix as follow: $\boldsymbol{U} \in \mathbb{R}^{N \times d}$,$\boldsymbol{S} \in \mathbb{R}^{d \times d}$, and $\boldsymbol{V} \in \mathbb{R}^{d \times D}$.

The key intuition behind the successfully working of PCA lies in the idea of use of covariance matrix, as covariance matrix contains the estimates of how each features in $\boldsymbol{X}$ related to every other features $\boldsymbol{X}$. Each eigenvector represents the direction in which the data can be dispersed to maximize the variance in the latent space. Finally, the eigenvectors captures the importance of each of these directions, i.e. the eigenvectors.

Some of the key properties of eigenvectors learned using PCA:

- **Orthogonal**: any two eigenvectors are perpendicular to each other, i.e. $U_i.U_j = 0$ thus formed basis vectors
- **Normal**: any eigenvector in the $\boldsymbol{U}$ or $\boldsymbol{V}$ is a normal vector, i.e. $|\boldsymbol{U}_i| = 1$

Therefore, eigenvectors are referred as *orthonormal* vectors.

2.2.2 Infinite Feature Selection (InfFS)

With the increase in the dimensionality of the data, feature selection has become a prominent pre-processing step for many machine learning and deep learning models. The crux of the feature selection/ranking methods is to identify set of strong features that are not redundant and irrelevant.

Infinite Feature Selection (InfFS) is an unsupervised feature ranking method. InfFS transforms the problem feature ranking in graph centrality problem [13]. Here, relationship between each pair of features is defined in terms of there statistical properties and mutual correlation. The goal here is to maximizing the divergence amongst the features and ranking the features high that are most dissimilar to there counter part and carry most self divergence, i.e. standard deviation.

Specific algorithmic details is presented in the Algorithm 1. This section will discuss the intuition behind various components in the InfFS. As noted earlier, InfFS, view the feature ranking problem as creating an affinity graph, and subset of feature paths connecting them. The cost of the path between two features is the defined in terms of their variance and correlation.

InfFS creates a graph, $\mathcal{G}\ (\mathbb{V}, \mathbb{E})$, such that, $\mathbb{V}$ represents the features in input matrix, $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, and $\mathbb{E}$ represents the pair of two features. The graph can be considered as an adjacency matrix $\boldsymbol{A}$, where each element $A_{ij} \in \boldsymbol{A}, 1 \leq i, j \leq D$ is defined as:

$$A_{ij} = \alpha\sigma_{ij} + (1 - \alpha)C_{ij}, \tag{2.21}$$

here, $\alpha \in [0, 1]$ is the loading coefficient, σ_{ij} being the maximum standard deviation among the two features, $X^{(i)}$ and $X^{(j)}$, s.t. $\sigma_{ij} = max(std(X^{(i)}), std(X^{(j)}))$, and C_{ij} is defined as $C_{ij} = 1 - |spearman(X^{(i)}, X^{(j)})|$ and represents the inverse Spearman's rank correlation coefficient i.e. . thus, A_{ij} is measure to describe the

Algorithm 1: Infinite Feature Selection [97]

Data: $\boldsymbol{X} = \{X^{(1)} \dots X^{(D)}\}$, α

Result: $\tilde{s}$ energy scores, for each feature

Building the graph

for $i \leftarrow 1$ **to** D **do**

  for $j \leftarrow 1$ **to** D **do**

    $\sigma_{ij} = max(std(X^{(i)}), std(X^{(j)}))$

    $C_{ij} = 1 - |spearman(X^{(i)}, X^{(j)})|$

    $A_{ij} = \alpha\sigma_{ij} + (1 - \alpha)C_{ij}$

  end

end

Letting paths tend to infinity

$r = \frac{0.9}{\rho(\boldsymbol{A})}$

$\boldsymbol{S} = (\boldsymbol{I} - r\boldsymbol{A})^{-1} - \boldsymbol{I}$

$\tilde{s} = \tilde{S}e$

return $\tilde{s}$

maximal feature dispersion and their correlation.

In order to rank the features of their importance, InfFS proposed to measure the importance in terms of path between two node in the graph. To do, so they leverages the Katz centrality [58] to measure betweenness centrality, as follows:

$$\boldsymbol{S} = (\boldsymbol{I} - r\boldsymbol{A})^{-1} - \boldsymbol{I} \tag{2.22}$$

Finally, the overall feature score is defined as

$$\begin{aligned} &\forall i = 1 \dots D \\ &\tilde{s_i} = \sum_{j=0}^{D} S_{i,j}. \end{aligned} \tag{2.23}$$

2.3 Deep Features

Deep features, also referred to as deep representation in the literature, can be seen as a response of the neuron or the layer (as a whole – set of neuron) in a neural network. The term "*deep*" refers to the position of the layer or neuron in the network structure, and the term "*feature*" refers to the neuron/layer output. In order to contain deep features, input data (with minimal pre-processing) is passed through the neural network, $\mathcal{N}$. A neural network can be defined as a sequential arrangement of trainable layers, such as convolution, recurrent and fully connected layers. Layers, such as convolution and recurrent, acts as a feature extractor and the layers, such as fully connected, acts as a classifier (classification problem) or predictor (regression problem). Most widely used neural network training approach is a supervised learning, therefore, the deep features can be seen as a generalized representation of the each sample in the input data. It is further observed that as the depth of the network increases, each layer/neuron capture information (representation) at different scale, i.e. the more shallow the layer is more finer details are captured, such as edges in patterns, intensity of pixels, however, more deeper one goes, the captured features become more abstract i.e. general shape of the object in the image. Therefore, deep features differ as different depths in the network.

For the purposes of this dissertation, the focus is on the overall network output rather then exploring different deep features at different depths in the network. In remainder of this dissertation, a neural network is considered as a sequential arrangement of linear (type of neuron/layer) and non-linear operations (neural activation) that aim at learning a mapping function, f , that maps the input data $\boldsymbol{X}$ to output $\boldsymbol{Y}$ s.t.

$$f : \boldsymbol{X} \rightarrow \boldsymbol{Y}. \tag{2.24}$$

As pointed out by Bergstra [10], the ultimate objective of the neural network is to minimize the expected loss/model error, E , over i.i.d. X samples of the learning function, f representing the network, $\mathcal{N}$. Often the success of the learning function, f, depends on the choice of hyper-parameters, θ .

$$f : \boldsymbol{X} \rightarrow \boldsymbol{Y} \mid \theta. \tag{2.25}$$

A typical neural network contains variety of heterogeneous components that constitutes as network hyper-parameters, and to learn multiple network parameters simultaneously, such as:

- **Input layer** is the first layer in the network that acts as a gateway to the deep network to accept raw or pre-processed in inputs for the network to feed on
- **Trainable layers** are the layers that contains the majority of the network that are learned during model training.This layer is generally represented as $\boldsymbol{Y} = \boldsymbol{W}\boldsymbol{X} + \boldsymbol{B}$, here $\boldsymbol{W}$ and $\boldsymbol{B}$ are the network weights and bias matrix
- **Activation layer** provides the non-linearly to the network to model a generalized representation for complex non-linearly separate data samples
- **Pooling layer** allows us to control the amount of intermediate data that needs to propagate through the network. As the depth increases, the amount of data (floating point operations) in the network increases. Therefore, the pooling layer allows to down-size the data as the depth increases
- **Fully-Connected layer** often referred to as FC, is the final output layer of the network where the target is situated, for classification the number of classes is the size of the FC layer with "*softmax*" activation and for regression problem, number of target variables with "*linear*" activation defines the FC layer

Chapter 3

RACKNet: ROBUST ALLOCATION OF CONVOLUTIONAL KERNELS

3.1 Overview

Deep neural networks have demonstrated unprecedented success in various applications involving multimedia objects, including images, text, and more complex signals. However, these networks are complex, with a large number of hyper-parameters. Despite their impressive success when these hyper-parameters are suitably fine-tuned, the design of good network architectures remains an art-form rather than a science: while various search techniques, such as grid-search, have been proposed to find effective hyper-parameter configurations, often these parameters are hand-crafted (or the bounds of the search space are provided by a user). This chapter argues, and experimentally show, that one can minimize the need for hand-crafting, by relying on the dataset itself. In particular, RACKNet show that the dimensions, distributions, and complexities of localized features extracted from the data can inform the structure of the neural networks and help better allocate limited resources (such as kernels) to the various layers of the network. To achieve this, RACKNet observes present several hypotheses that link the properties of the localized image features to the CNN and RCNN architectures and then, relying on these hypotheses, present a RACKNet framework[1] which aims to learn multiple hyper-parameters by extracting information encoded in the input datasets. Experimental evaluations of RACKNet against major benchmark datasets show that

[1] Garg, Yash, and Candan, K. Selçuk. RACKNet: Robust Allocation of Convolutional Kernels in Neural Network for Image Classification. International Conference on Multimedia Retrieval, 2019.

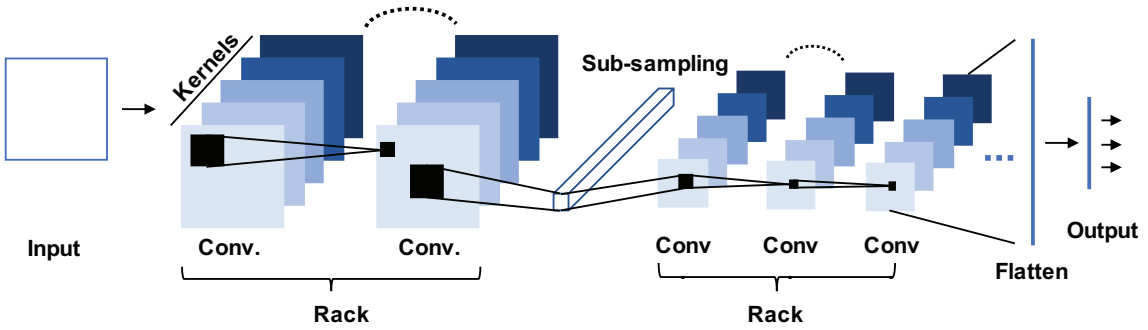


Figure 3.1: Outline of a CNN (in this Chapter, Set of Convolutional Layers Between Two Pooling Layers is Referred as “Racks”)

RACKNet provides significant improvements in the network design and robustness to change in the network.

3.2 Introduction

Deep neural networks, including convolutional neural networks (CNNs, Figure 3.1) have seen successful application in face recognition [64] as early as 1997, and more recently in various multimedia domains, such as time series analysis [130, 121], speech recognition [47], object recognition [70, 102, 101], and video classification [57, 105].

More recently, CNNs’ successful application in a variety of multimedia domains has lead to a shift away from feature driven algorithms, such as SURF [8], HOG [29], and SIFT [76], into the design of well-crafted CNN architectures for specific datasets and application domains. Unfortunately, deep neural networks, including CNNs, tend to be complex with a large number of hyper-parameters. As [10] points out, the ultimate objective of finding a high performing architecture configuration to minimize the expected loss, $L(x; l_f)$, over i.i.d x samples of the learning function, l_f, representing the network, NN. Often the success of the learning function, l_f, depends on

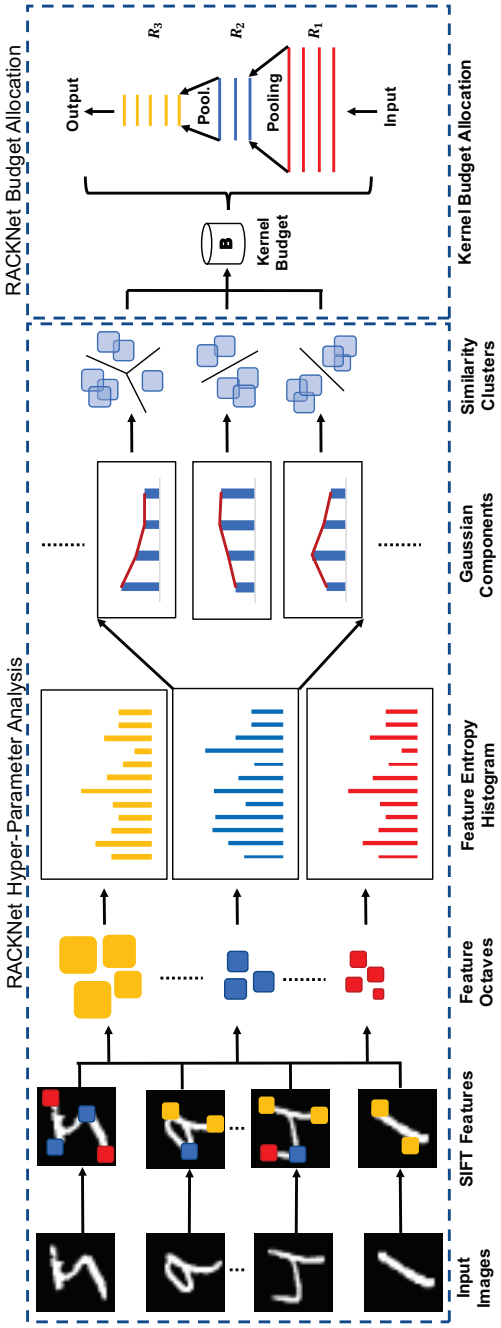


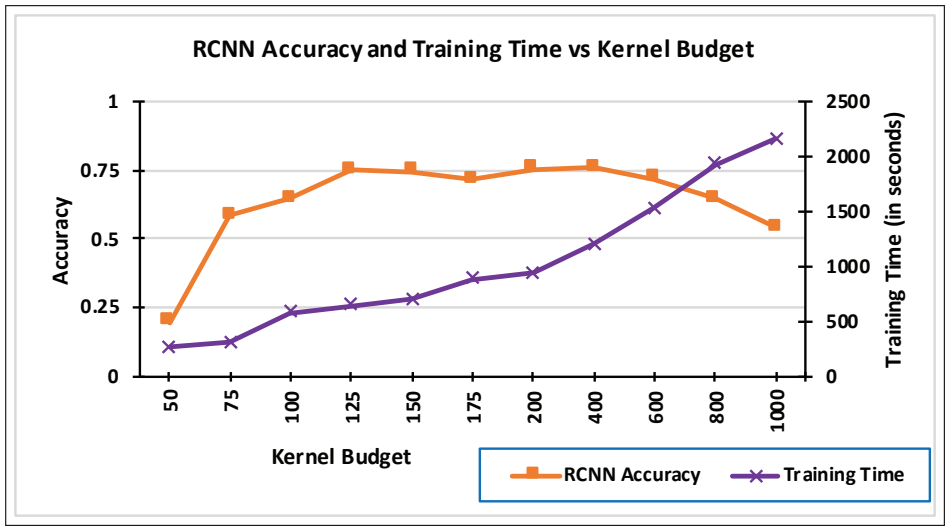
Figure 3.2: Overview of the RACKNet Framework: Sizes, Complexities, and Distributions of the Local Features Extracted from the Image Dataset During Pre-Processing are Used to Inform the Structure of the CNN and Allocate Convolution Kernels Within the Neural Network Architecture

the choice of *hyper-parameters*, λ. Therefore, L is a function of hyper-parameters as well, where $L(x, \lambda; l_f)$. Despite their impressive success when these hyper-parameters are suitably fine-tuned, design of good network architectures still remains an art-form rather than a science, while various techniques, such as random search [11, 10], grid-search [63], and others [37, 108, 26, 55, 107, 131], have been proposed to help locate an effective (optimal or close-to-optimal) hyper-parameter configuration, λ_o. Due to high-dimensionality of the hyper-parameter space and the complexity and non-linearity of the CNN architectures, searching for an effective hyper-parameter configuration, λ_o, is a computationally-expensive process, which has led to an interest in specialized and targeted approaches [37, 108, 125, 80, 55] that introduce refinements *on-top* of existing high performing network configurations. Yet, today these parameters need to be hand-crafted (or at least the bounds of the search space are provided by a user).

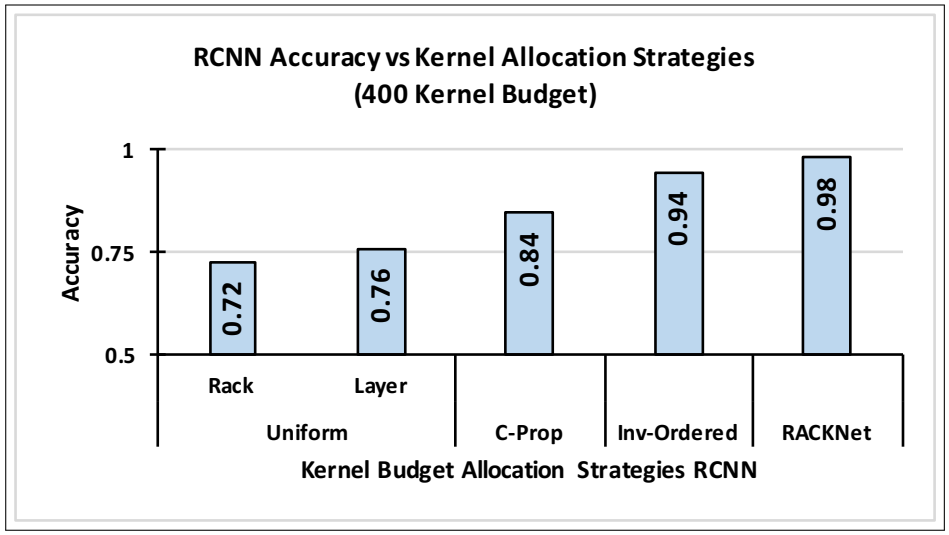
As further discussed in Section 3.3, a common shortcoming of the existing approaches to hyper-parameter search is that they "*work on the data, not with the data*", i.e. they aim to find a hyper-parameter configuration that minimizes L, but in the process they ignore the available data and the key insights that the data can provide in honing in on effective configurations of hyper-parameters. In contrast, RACKNet argues that one can minimize the need for hand crafting of the search space, by relying on a pre-training analysis of the data itself. To this end, this chapter presents a RACKNet framework which aims to learn key hyper-parameters by extracting information encoded in the input datasets (Figure 3.2).

3.2.1 Key Contributions

RACKNet focuses on hyper-parameters that impact kernel budget allocation, which (as one can see in Figure 3.3) can have significant impacts on model accura-



(a) Impact of Kernel Budget on Accuracy and Training Time



(b) Impact of Kernel Allocation Strategies on Model Accuracy

Figure 3.3: Impact of Kernel Budget Allocation on Accuracy and Training Time (More Details in Section 3.5)

cies and training times. In particular, RACKNet show that the sizes, complexities, and distributions of localized features, such as SIFT features [76], extracted from the dataset can provide insights that can inform the structure of the neural networks and help better allocate limited resources (such as kernels) within the network.

> In other words, RACKNet argues that, even when the SIFT features may not be sufficiently informative to achieve high accuracies in media analysis tasks alone, they can provide reliable insights that can inform the design of effective CNN and RCNN architectures.

More specifically, unlike purely feature-based approaches, which leverage features extracted from the data to implement the analysis task, RACKNet uses these features only to inform the structure of the CNN (or RCNN) to be used for analysis. To achieve this, RACKNet first present four key observations that link the properties of the localized features to the CNN and RCNN architectures:

- *Observation 1:* Sizes of the localized features in the dataset can inform the kernel sizes and the numbers of sub-sampling racks of the CNN.
- *Observation 2*: Complexities of the features (measured through the entropies of their descriptors) at different scales can be used to discover the number of layers per rack.
- *Observation 3*: Overall complexities of convolutional layers, defined through corresponding image features, can be used to inform the arrangement of layers within a rack.
- *Observation 4*: Distribution of the features for a given complexity component can be used to discover the number of kernels per convolution layer.

It is important to note that, while RACKNet uses both localized and CNN features, they are used for entirely different purposes:

- cheaper-to-obtain, but rough, localized features are used to bootstrap the hyper-parameter of the CNN; whereas
- expensive-to-obtain, but finer-grain, CNN features are, then, used to obtain the classifier.

RACKNet is designed as an unsupervised general purpose single-shot architecture search framework that takes pre-computed localized image features for hyper-parameter extraction (as described in Section 3.4). RACKNet argues, and experimentally show, these feature can provide insights on the dataset that can inform the architecture of the network. RACKNet is evaluated on common benchmark datasets, such as MNIST, SVHN, CIFAR10, COIL20, and ImageNet, and a commonly used localized image feature, known as SIFT [76]. Experiments on these datasets show that RACKNet indeed provides significant improvements in the network performance.

3.2.2 Organization of the Chapter

This chapter is organized as follows: Section 3.3 describes the existing work in detail, Section 3.4 presents the proposed RACKNet framework, and Section 3.5 discusses the performance and robustness of RACKNet framework under various settings involving change in kernel budget and dropout rate for both CNN and RCNN designs. Chapter conclude in Section 3.6.

Table 3.1: Key Notations Used in this Chapter (More Details in Appendix B)

Symbol	**Convolutional Neural Network (CNN)**	Symbol	**Scale Invariant Feature Transform (SIFT)**
r	# of racks of conv. layers	o	# of Octaves
c_i	# of conv. layers in the given rack,R_i	g_i	# of Gaussian comp. in entropy histogram an octave, o_i
$k_{i,j}$	# of kernels in a given conv. layer, $C_{i,j}$	$n_{i,j}$	# of feature-similarity clusters in a given Gauss. comp., $g_{i,j}$
$e_{i,j}$	Kernel size for a given conv. layer, $C_{i,j}$	$\sigma_{i,j}$	Feature scope in a given Gauss. comp., $g_{i,j}$

3.3 Related Work

Successful application of convolutional neural networks (CNN) dates back to the 90s where it was applied to face recognition [64]. More recently CNNs have shown promising results in various multimedia applications such as image enhancement [17], speech recognition [47], video classification [57, 105], object recognition [70, 102, 101], and time series analysis [130, 121] have generated significant interest in CNN deisgn and configuration.

[63, 11, 10] proposed hyper-parameter configuration search techniques that evaluate performances of different configurations on training data. [10], for example, implemented a random search over a bounded 32-dimensional parameter space. [131] proposed a reinforcement learning based technique to incrementally improve hyper-parameter configurations. [26] proposed an evolutionary search algorithm that generates a large population of CNNs to converge on a good configuration through mutations. [53] designed a greedy search through multi-attribute learning.

The main difficulty with these search-based techniques is that the process can be very costly as the possible parameter space can be very large. The prohibitive cost of search-based approaches led to approaches that target specific network components; these include maxout [37], batch normalization [55], rectified-lu [62,

80, 45], and dropout [108], techniques: [37] proposed a *maxout* strategy that introduces a max-pooling layer that provides spatial invariance to the network. [55] proposed a batch normalization layer aiming to minimize the covariate shift in the network, allowing for higher learning rates. The *dropout* layer proposed in [108] aims to prevent overfitting of the network by randomly engaging and disengaging the convolutional kernels during training. [62] proposed Rectified Linear Units (ReLUs) aiming to minimize the loss of gradients during the training phase of the network. [80] proposed a non-linear rectifier that reduced the sparsity in the network. [45] proposed Probabilistic-ReLU, advancing the conventional ReLUs, to adaptively learn the ReLU parameters. Fusion networks have been proposed that combine shallow and deep features [16, 129].

This RACKNet notes that a major weakness of the various techniques discussed above is that they ignore the input data itself. In particular, RACKNet argues that it may be possible to use the training dataset itself to help search for appropriate CNN hyper-parameter configurations. While this data-driven approach has not been used in CNN design, RACKNet observes that it found successful use in several other application domains. For example, [127] demonstrated that spatio-temporal features extracted from time series data can reduce the computational cost of determining warping paths between multi-variate time series. [104, 72] leveraged spatio-temporal features extracted from multi-variate time series to improve on time series classification. [34, 31] leveraged metadata extracted from datasets to improve classification accuracies. This chapter further notes that even when the local image features alone may not be able to provide high accuracies in media analysis, they can provide insights that can inform the design of effective CNNs.

3.4 Robust Allocation of Convolution Kernels for CNNs (RACKNet)

As discussed above, while well designed CNNs can be very effective in image classification, their effectiveness is often hampered by the need for hand-crafted hyper-parameter design, especially for new datasets that have not been seen in the past. In this section, RACKNet framework is proposed that bootstraps the hyper-parameter configuration for CNNs based on a pre-analysis of the image dataset, $\mathcal{D}$, (Figure 3.2). In particular, in order to better distribute the given budget, B, of convolutional kernels across CNN layers, RACKNet proposes to leverage SIFT features that capture local image patterns. RACKNet translates the sizes, complexities, and distribution of the high-level feature patterns in the given dataset to hyper-parameters of CNN structures that best utilize the given kernel budget.

3.4.1 Hyper-Parameters of a CNN

A convolutional neural network (CNN [65]) is a type of neural network that works by leveraging the local spatial arrangements by establishing local connections among small spatial regions across the adjacent layers. A CNN consists of several complementary components organized into layers (Figure 3.1):

- Each *convolution* layer links local-spatial data (i.e., pixels at the lowest layer) through a set of filters or *kernel*s that represent the local spatial features identified in the data.
- Since each convolution layer operates on the output of the previous convolution layer, higher layers correspond to increasingly complex features obtained by combining lower-complexity features.
- Since relevant features of interest can be of different sizes, *pooling/subsampling*

layers are introduced among convolution layers: these pooling layers carry out down-sampling of the output of a convolution layer, thereby (given a fixed kernel size) effectively doubling the size of the feature extracted by the corresponding filter.

Therefore, intuitively, a CNN searches for increasingly complex local features that can be used for understanding (and interpreting) the content of a dataset. Given a dataset, and labeled data, this is achieved by a process known as *back-propagation* that uses gradient-search to identify the filters that best fit the labeled data.

Before the *back-propagation* can be implemented, however, one needs to pick hyper-parameters, such as the number of convolution layers, type of pooling operations, and number of kernels. In this chapter, RACKNet follows the following convention (see also Table 3.1):

- RACKNet divides the sequence of convolution layers into r *racks*, each corresponding to a different feature size: to enable this, the two consecutive racks of convolution layers are separated by a pooling/subsampling layer.
- For each rack, R_i, of convolutional layers, RACKNet associates c_i many convolution layers, each resulting in more and more complex features of the size corresponding to the rack, R_i.
- Each convolution layer, $C_{i,j} \in R_i$, has $k_{i,j}$ kernels, each of size $e_{i,j} \times e_{i,j}$, where $e_{i,j}$ is the edge length of the kernel.

Therefore, the search for the hyper-parameters of the CNN can be posed as searching for the appropriate r, c, $k_{i,j}$, and $e_{i,j}$ parameter values that best describes the data.

3.4.2 Background: Local Image Features

As shown in the introduction, a common shortcoming of the existing approaches to hyper-parameter search is that they "*work on the data, not with the data*", i.e. they ignore the available data and key insights that the data can provide in honing in on effective configurations of hyper-parameters. In contrast, RACKNet argues that one can minimize the need for hand crafting of the search space, by relying on a pre-training analysis of the data itself. More specifically, RACKNet observes that if one could cheaply extract localized features of a given dataset (independent of the labeled data) – even if these features may not be sufficiently effective in achieving high accuracies in media analysis – the sizes, complexities, and distributions of these localized features can provide reliable insights that can inform the design of effective CNN architectures.

There are several localized feature extraction algorithms for images: these include SURF [8], HOG [29], and SIFT [76]. Scale Invariant Feature Transform [76], in particular, has been the *de facto* image representation strategy for content-based image retrieval as these features have shown robustness against rotation, scaling, and various distortions. SIFT [76] extracts stable and scale invariant patterns contained in a given image through a multi-step approach. SIFT supports multi-scale feature extraction: intuitively, SIFT features correspond to regions in a given image that are *different* from their neighborhoods, also in different image scales. Consequently, starting from the smallest feature size (provided by the user), features are organized into *octaves* corresponding to doubling of the feature diameter. In the rest of this section, it is shown that sizes and complexities of the localized features in a given dataset can be used to inform the design of CNNs that will operate on that dataset.

3.4.3 Observation #1: Number of Racks

As described earlier, RACKNet organizes the layers of a CNN in the form of racks of convolution layers, where each rack corresponds to features of a different size: in particular, the down-sampling (through average pooling) operation between two consecutive racks reduces the number of rows and columns by half. Therefore, it is easy to see that there is a correspondence between the number of racks, r, of a CNN and the number of octaves, o, of localized features extracted from images in a given dataset.

Note that, while in general the number of octaves is a user provided input parameter to the SIFT algorithm, in general the number of features the algorithm identifies drops with increasing octaves. This is because, while an image may contain many small size features, the number of large yet stable features declines with the feature size. Therefore, the first hypothesis is that

$$r = o_{\mathcal{D}},$$

where $o_{\mathcal{D}}$ is the number of octaves where one is able to detect sufficiently many features in a given dataset, $\mathcal{D}$.

3.4.4 Observation #2: Number of Convolution Layers in a Rack

As discussed in Section 3.2, each convolution layer corresponds to a set of features (represented by kernels) of a different complexity [103].

> In other words, the more diverse the complexities of image features of a given size, the more convolution layers the corresponding rack needs to have.

Therefore, the next observation is that one can determine the number of convolution

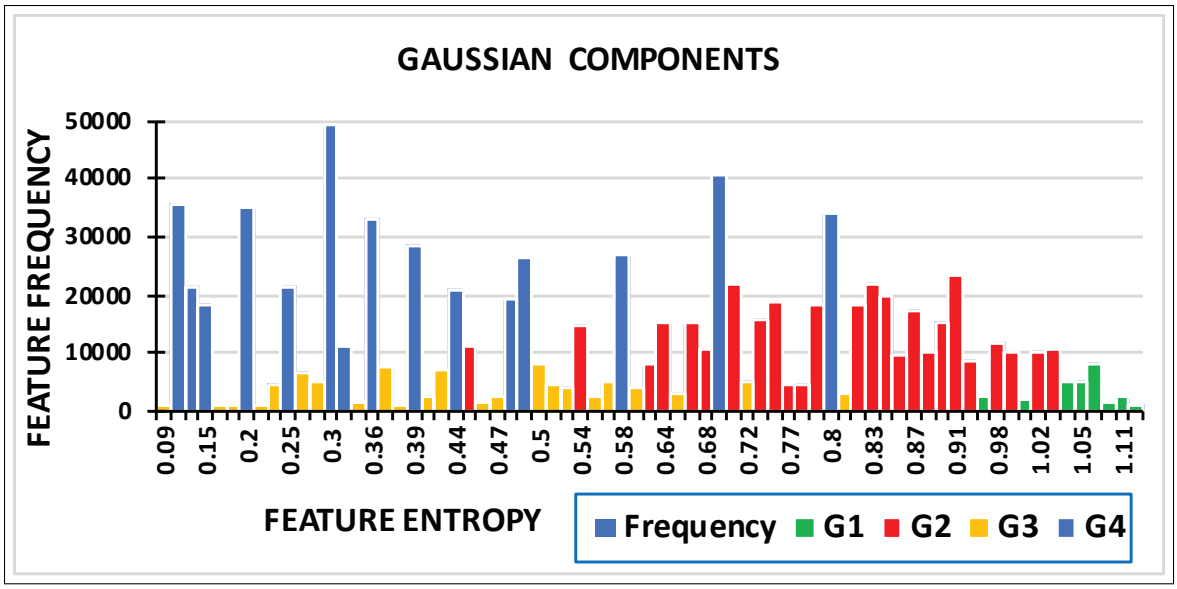


Figure 3.4: (Sample) Feature Entropy Distribution for the MNIST Dataset [66]: Here, Different Colors Correspond to Four Different Gaussian Components Identified in this Complexity Histogram (Figure Best Viewed in Color).

layers in a given rack, by analyzing the distribution of the complexities of the SIFT features in the corresponding octave, extracted from the dataset.

3.4.4.1 Feature Complexity and Distribution

In this chapter, RACKNet proposes to measure the complexity of a SIFT feature, F, in terms of the entropy of the pixels within the corresponding scope as

$$E(F) = -\sum_{a \in pixel\_amplitudes} P_{scope(F)}(a) log P_{scope(F)}(a),$$

where, $pixel\_amplitudes$ is the set of possible pixel values and $P_{scope(F)}(a)$ is the distribution of the amplitude value a within the scope of the feature F. Note that each SIFT feature, F, has a center $\langle x_F, y_F \rangle$ and a scale σ_F represent the amount of Gaussian smoothing corresponding to the feature. Since under Gaussian smoothing, 3 standard deviations would cover $\sim 99.73\%$ of the pixels that have contributed

to the identified feature, the radius of a feature F is defined as $3\sigma_F$. To compute $P_{scope(F)}(a)$, all pixels are considered within the $\pm 3\sigma_F$ from $\langle x_F, y_F \rangle$ in either of the two directions. Next a complexity histogram, $\mathcal{H}(\mathcal{D}, O_i)$, describing the entropy distribution of the features extracted from the dataset $\mathcal{D}$, corresponding to octave O_i, is computed.

3.4.4.2 Number of Convolution Layers

As seen in Figure 3.4, the entropy distribution is rarely uniform and shows a certain degree of clustering. Therefore, it can be argued that the entropy distribution can be leveraged to determine the number of convolution layers in a given rack. In particular, complexity histogram can be treated as a mixture of k (possibly overlapping) Gaussians, where each Gaussian component corresponds to a distinct feature's complexity. Thus, given the entropy histogram, $\mathcal{H}(\mathcal{D}, O_i)$, corresponding to octave, O_i, a non-parametric Gaussian mixture separation algorithm[2] can be used to (a) identify the number, g_i, of Gaussian components and (b) to associate each distinct entropy instance to one of these components. Thus, the number of convolutional layers can be determined as

$$\forall_{i=1...r}\ c_i = g_i.$$

where, c_i is the number of convolution layers corresponding to the i^{th} rack using the number of Gaussian components„ g_i of the corresponding octave, O_i.

3.4.5 Observation #3: Organization of Convolution Layers within a Rack

As discussed in Section 3.4.1, during the *feed forward* phase of CNN, the complexities of the patterns learned increases as the training process moves from one

[2] In the implementation, the Gaussian mixture model available through the Python library, scikit-learn [93] is used.

convolutional layer to another.

> Therefore, the local image features extracted from the image dataset should be mapped to the convolution layers in the order implied by the Gaussian components to which they belong.

More specifically,

$$\forall_{i=1...r} \forall_{j,h=1...c_i} \quad (j > h) \leftrightarrow (\mu_{i,j} > \mu_{i,h}) \,,$$

where $\mu_{i,j}$ is the mean of the j^{th} Gaussian component of the i^{th} rack. As described next, this order of complexities is used in distributing the kernel budget to the convolution layers in a rack.

3.4.6 Observation #4: Number of Kernels in a Convolution Layer

The next key observation is that:

> the number of kernels corresponding to j^{th} convolution layer of the i^{th} rack should reflect the number of *relevant* distinct patterns of the corresponding feature size and complexity.

Unfortunately, without access to the labeled data, there exists no information about the number of relevant distinct patterns. However, as shown in in next section, one can replace this constraint with a more relaxed constraint which can be readily computed.

3.4.6.1 Convolution Layers with Non-Uniform Kernel Counts

Let $G_{i,j}$ be one of the g_i Gaussian components obtained in the previous step through the analysis of the complexity histogram at octave O_i. Let $\mathcal{F}_{i,j}$ be the corresponding set of SIFT features, again identified as a by-product of the non-parametric

Gaussian separation process. RACKNet argues that the number, $k_{i,j}$, of kernels corresponding to the j^{th} convolution layer of the i^{th} rack should be *inversely proportional* to the number of similarity clusters[3] for the features in $\mathcal{F}_{i,j}$:

$$\forall_{i=1...r}\forall_{j=1...g_i}\ k_{i,j} \overset{inv}{\sim} num\_clusters(\mathcal{F}_{i,j}).$$

While this initially sounds surprising, there is a simple explanation for this relationship between the number of feature similarity clusters and the number of kernels:

> the higher the number of clusters of $\mathcal{F}_{i,j}$ one can identify, the more distinguishable the underlying feature patterns are and thus, the fewer the number of kernels are needed to distinguish these patterns.

This observation is validated in Section 3.5 (Table 3.6). Note that given this observation and given a total budget of B kernels for the entire CNN, the number, $k_{i,j}$, of kernels corresponding to the j^{th} convolution layer of the i^{th} rack is computed as

$$\forall_{i=1...r}\forall_{j=1...c_i}\ k_{i,j} = \beta_{i,j} \times B,$$

and

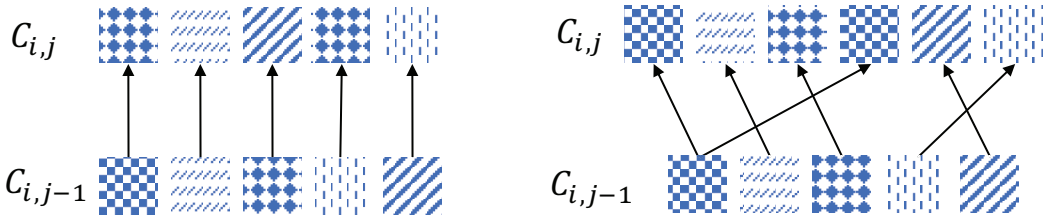
$$\beta_{i,j} = 1 - \left(\frac{c_i}{\sum_{p=1...r} c_p} \times \frac{n_{i,j}}{\sum_{l=1...c_i} n_{i,l}} \right)$$

where, $n_{i,j} = num\_clusters(\mathcal{F}_{i,j})$ is the number of descriptor similarity clusters for octave O_i and Gaussian component, $G_{i,j}$.

3.4.6.2 Weight Sharing with non-Uniform Kernel Counts

An important decision in network design is the connectivity among kernels across consecutive layers. This has two aspects to consider: (a) how convolutions at a

[3] In the implementation, the non-parametric density-based spatial clustering algorithm DBSCAN [28] with cosine distance to identify the number of similarity clusters, $n_{i,j}$, of the descriptors of the features in $\mathcal{F}_{i,j}$ is used.



(a) Conventional Weight Sharing

(b) Proposed Weight Sharing

Figure 3.5: Weight Sharing: in Conventional Strategy, Weight Sharing is on one-to-one Basis; in the Proposed Approach, 1-M/N-1 Sharing is Possible Based on Kernel Similarities.

down-stream layer are related to the convolution results of the up-stream layer and (b) how, during training, weights can be shared across kernels in these two layers (or how weights learned for kernels in one layer are used to bootstrap the weight for the another consecutive layer). For instance,

- in terms convolution connections, [108] (a CNN implementation) and [70] (an RCNN implementation) both rely on full connectivity across consecutive convolution layers.
- in terms of weight sharing, however, [108] assumes no weight sharing across layers, whereas [70] assumes that weight sharing occurs on a one-to-one basis between corresponding kernels in two layers: i.e., during the feed-forward stage of each iteration, the weight for a downstream kernel is initialized with the weight of the corresponding kernel in the up-stream layer.

Note that weight sharing described above is possible because the number of kernels across two consecutive layers are the same. As shown in the Section 3.4.6.1, in RACKNet, there may be different number of kernels across consecutive layers; this means that, when weight sharing is implemented, a one-to-one strategy will

not work. This difficulty can be overcome by mapping each $K_{i,j,l}$ (in the j^{th} convolution layer of the i^{th} rack) to the most similar kernel, $K_{i,j-1,h}$, s.t. $\forall_{i=1...r}, \forall_{j=2...g_i}$, and $\forall_{l=1...k_{i,j}}$, consequently

$$\mu(K_{i,j,l}) = K_{i,j-1,h} \text{ s.t. } h = \underset{h'=1...K_{i,j-1}}{argmax} \left\{ cos(\vec{K}_{i,j-1,h'}, \vec{K}_{i,j,l}) \right\}.$$

Here $\mu()$ corresponds to the mapping function and $\vec{K}_*$ corresponds to the vector representations of the convolution patterns learned during the training of the network. Note that, since the discovered patterns change at each iteration, the mapping $\mu()$ needs to be revised accordingly. Note also that the kernel alignments learned during the feed-forward stage are preserved and leveraged during the back-propagation of the gradients as well. In particular, during back-propagation, gradients are redirected to the corresponding kernel mappings, $\mu()$, for synchronous error correction.

3.5 Experiments

In this section, the proposed RACKNet framework is evaluated for data-driven CNN design and compare its classification accuracy performance against alternative schemes.

3.5.1 Experimental Setup

RACKNet was implemented RACKNet in Python environment using TensorFlow framework [2] and Keras [22]. During the training of CNN, 10% of the training data was reserved for validation to evaluate model quality. Root Mean Square Error (RMSE) is used as the model optimizer. *rectified-linear unit* (ReLU) [80] and *softmax* [14] as the hidden and output activation function, and for pooling *average* or *maxout* [37] were used. Default kernel budget of 400 anddefault dropout rate of

0.25 is used. MatLab implementation was used to extract SIFT features [76], and scikit-learn [93] both to search for Gaussian components of entropy histograms, and for feature similarity clustering using DBSCAN. All experiments were executed on an Intel Xeon E5-2670 2.3 GHz Quad-Core Processor with 32GB RAM[4].

3.5.2 Competitors

The proposed framework is compared against several competitors: *wide net* [4], *maxout* [37], *dropout* [108], and RCNN [70]. Furthermore, to assess the usefulness of the four key observations that form the core of RACKNet, several alternative kernel allocation strategies were considered:

- **Random budget allocation** distributes the kernel budget, B, to the conv. layers at random, $s.t.\ \sum_{\forall_{i=1\ldots r}} \sum_{\forall_{j=1\ldots c_i}} \beta_{ij} \times B = B$.
- **Uniform rack budget allocation** allocates the same number of kernels for each rack in the CNN and then uniformly allocates kernels for each layer in the rack; $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{1}{c_i} \times \frac{1}{r}$.
- **Uniform layer budget allocation** allocates the same number of kernels for each convolutional layer irrespective of the convolution layer to which it belongs; i.e., $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{1}{\sum_{p=1}^{r} c_p}$.
- Section 3.4.6 showed that RACKNet allocates kernels to convolution layers in a rack inversely proportional to the feature complexities. **C-Proportional allocation** strategy uses the opposite strategy and allocates kernels to convolution layers in a rack *directly proportional* to the feature complexities: $\forall_{i=1\ldots r} \forall_{j=1\ldots c_i}\ \beta_{ij} = \frac{c_i}{\sum_{p=1}^{r} c_p} \times \frac{n_{i,j}}{\sum_{p=1}^{c_i} n_{i,p}}$.

[4]Results presented in this chapter were obtained using "*Chameleon: A Large-Scale Reconfigurable Experimental Environment for Cloud Research*" (NSF Award No. 1743354)

Table 3.2: Convolutional Layers and Numbers of Kernels ('-' Denotes Downsampling Layer by 2) with Kernel Budget 400.

Datasets	**Convolution Layers** ⟨**Kernels**⟩
MNIST	4-4-4 ⟨35,36,32,32-30,36,30,40-35,31,35,35⟩
SVHN	3-4-3 ⟨39,30,53-35,39,37,40-40,40,41⟩
CIFAR	5-3-3 ⟨39,29,38,39,39-33,39,39-37,37,37⟩
COIL	2-4-4 ⟨8,77-44,29,44,44-46,25,46,46⟩
ImageNet	10-12-6 ⟨10,16,12,13,16,21,19,15,10,11-14,11,16,15, 17,12,15, 19,12,15,14,12-14,18,10,13,16,14⟩

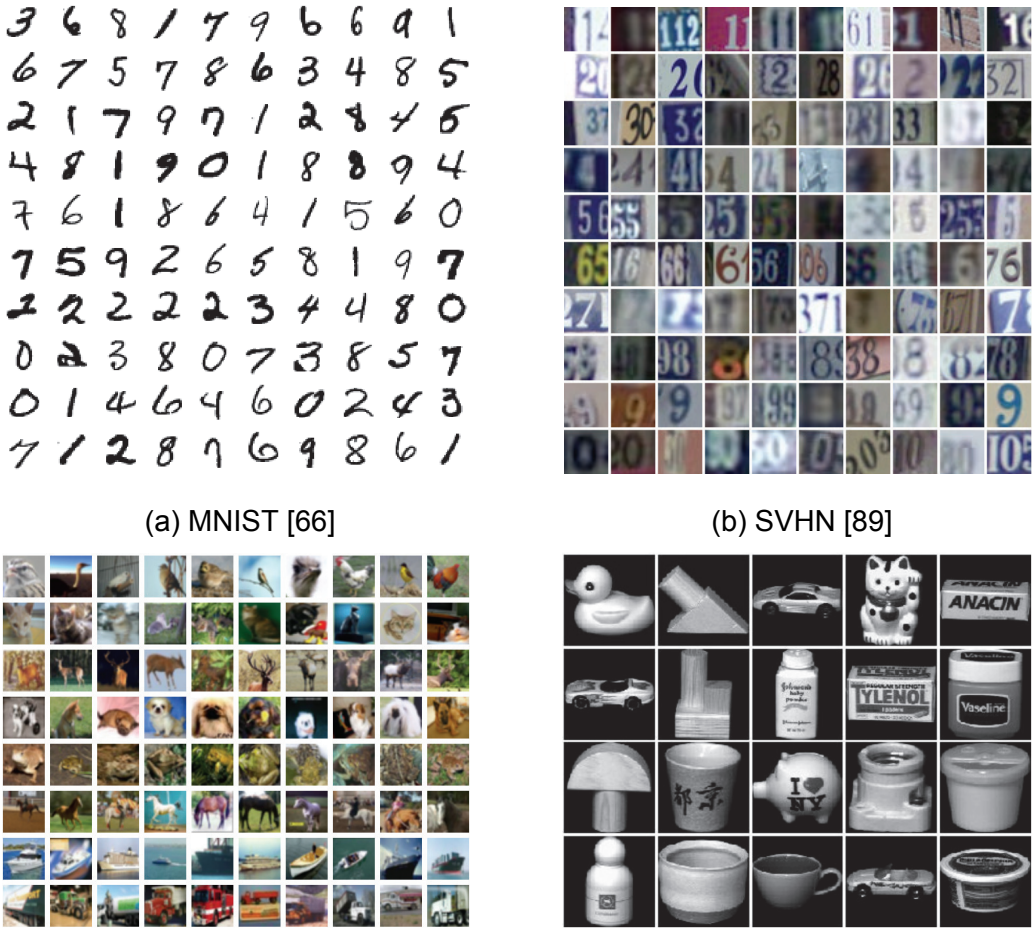
- In noted in Section 3.4.5, RACKNet orders convolution layers ibased on the feature complexities. **Inverse-order allocation** uses the opposite allocation strategy, and places convolution layers with higher feature complexities later in the *feed-forward* sequence.

In all scenarios, the number of racks and the numbers of convolution layers per rack are selected per Observations #1 and #2.

3.5.3 Benchmark Datasets

For evaluation, various commonly used benchmark datasets; including *digit* datasets, MNIST and SVHN, and *real-world image* datasets, CIFAR10, COIL20, and ImageNet (Figure 3.6) were considered.

- **MNIST** is a dataset containing $60k$ and $10k$ training and testing images, respectively of 28×28 handwritten digit captures (Figure 3.6(a)).
- **SVHN** dataset consists of 32×32 house numbers extracted from Google Street View images. The dataset consists of $73k$ and $26k$ images for train-



(a) MNIST [66]

(b) SVHN [89]

(c) CIFAR [61]

(d) COIL [88]

Figure 3.6: Samples from the benchmark datasets

ing and testing (Figure 3.6(b)).

- **CIFAR10** contains $50k$ training and $10k$ testing images, respectively, with 32×32 resolution and the dataset contains 10 labels (Figure 3.6(c)).

- **COIL20** contains images of 20 real-world objects. For each object, the dataset includes 72 images, captured at 5-degree intervals by rotating a turntable 360 degrees (Figure 3.6(d)).

- **ImageNet** contains $\sim$1.23 million images for 1000 real-world entities, with

Table 3.3: Entropy Histogram Bin Size vs Model Accuracy RACKNet (RCNN Implementation)

Bin Size	MNIST	SVHN	CIFAR	COIL
0.001	98.93	90.12	85.36	99.54
0.010	**99.48**	**97.85**	**95.74**	**99.86**
0.020	99.04	90.05	84.79	99.37
0.050	95.20	87.02	81.88	98.68

$\sim$1000 images per entity [62].

Table 3.2 shows the hyper-parameters extracted using the RACKNet. In particular, each feature octave corresponds to a "rack", and the number of layers per rack is determined using the data - not by user input. Entropy histogram bin size of 0.01 is used as default: as shown in Table 3.3, RACKNet performs well with this bin size, independent of the dataset.

3.5.4 Results

3.5.4.1 RACKNet vs. Competitors

In Table 3.4 and 3.5, the RCNN-based implementation of RACKNet is compared against various state of the art CNN and RCNN techniques. As the tables show, the network design based on the RACKNet framework provide the best overall accuracies, indicating that RACKNet allocates kernel resources more effectively than the handtuned competing architectures, with the exception of SENet [50] which has a similar accuracy to RACKNet. It is important to note that SENet which is significantly deeper than RACKNet, with 154 layers against only 28 layers wheras RACKNet achieves similar accuracies with as low as 390k hyperparameters as

Table 3.4: Reported Accuracies for the Competitors vs RACKNet (RCNN implementation) Accuracy.

	MNIST	**SVHN**	**CIFAR**	**COIL**
Maxout [37]	99.55	97.53	88.32	–
Dropout [108]	99.21	97.45	87.39	–
Deep Net [4]	99.07	–	–	99.23
Wide Net [4]	97.66	–	–	99.36
RCNN [70]	98.12	93.67	75.28	90.02
RACKNet	99.48	97.85	95.74	99.86
RACKNet-maxout	**99.72**	**98.54**	**97.62**	**100.00**

opposed to 26.9 million in SENet.

3.5.4.2 RACKNet vs. Alternative Allocation Strategies

Table 3.6 compares RACKNet-based kernel allocation to alternative kernel allocation strategies. Results in this table confirm that the four key observations that form the core of the RACKNet framework are highly effective and that RACKNet is the only strategy that consistently outperforms the random allocation strategy. It is especially important to note that while naïve allocation strategies, such as random, show reasonable performance on simple datasets, like MNIST and CIFAR10, they show very poor performance for complex ones, such as ImageNet, In contrast, RACKNet provides consistently superior performance for both simple and complex data sets.

Table 3.5: Top-1 Classification Accuracy for ImageNet

Approaches	**Parameters**	**Accuracy**
Dropout [108]	–	61.90
VGG-16 [106]	134M	72.70
PReLU [45]	–	78.41
Batch Normalization [55]	–	78.01
DenseNet-126 [52]	7M	74.98
RESNet-34 B [46]	0.46M	78.16
RESNet-34 C [46]	0.46M	78.47
SENet [50]	26.9M	**81.32**
Random	0.39M	69.12
Uniform-Layer (CNN-Plain)	0.39M	71.74
RACKNet-CNN	0.39M	76.82
RACKNet-RCNN	0.39M	79.14
RACKNet-RCNN-Maxout	0.39M	81.02

3.5.4.3 Kernel Budget, Dropout Rates, and Racks

In Figures 3.7 and 3.8, investigates the robustness of RACKNet (CNN and RCNN) against varying kernel budgets and dropout rates. As shown in the figures, RACKNet is robust against varying kernel budgets and dropout rates and can help provide significant protection against introduction of noisy kernels that degrade the network accuracy. Table 3.7 shows that RACKNet works well with 3 racks, as predicted by the number of feature octaves of localized SIFT features. Providing a higher number of racks does not contribute to performance as the number of SIFT features drop significantly for larger octaves, limiting any discernible insights from the cor-

Table 3.6: Accuracy for Various Kernel Allocation Strategies.

Strategies	Observation	MNIST	SVHN	CIFAR	COIL
Random	$\langle o_1, o_2 \rangle$	98.98	90.34	69.69	99.53
Uniform Rack	$\langle o_1, o_2 \rangle$	98.97	90.90	70.18	99.55
Uniform Layer	$\langle o_1, o_2 \rangle$	98.97	91.04	69.50	94.57
C-Proportional	$\langle o_1, o_2, o_3 \rangle$	98.90	90.77	68.50	99.16
Inverse Order	$\langle o_1, o_2, o_4 \rangle$	98.99	91.11	69.88	99.38
RACKNet	$\langle o_1, o_2, o_3, o_4 \rangle$	**99.25**	**94.57**	**71.65**	**99.94**

Table 3.7: Accuracy vs Number of Racks, RACKNet-RCNN

	MNIST	SVHN	CIFAR	COIL
Rack–2	99.56	98.51	97.31	**100**
Rack–3	**99.72**	**98.54**	**97.62**	**100**
Rack–4	98.78	97.21	96.83	99.63

responding features.

3.5.4.4 Execution Time

RACKNet requires extraction and analysis of local image features for their sizes and complexities before the CNN is trained. Table 3.8 presents the time cost of this process along with the CNN training time. As this table shows, the pre-processing overhead of RACKNet is not high. Since the pre-processing cost does not grow as fast as the CNN training cost, RACKNet becomes cost efficient especially for large training sets, such as SVHN. A key advantage of advantage of RACKNet over other hyper-parameter search approaches is that, while they have to train **multiple**

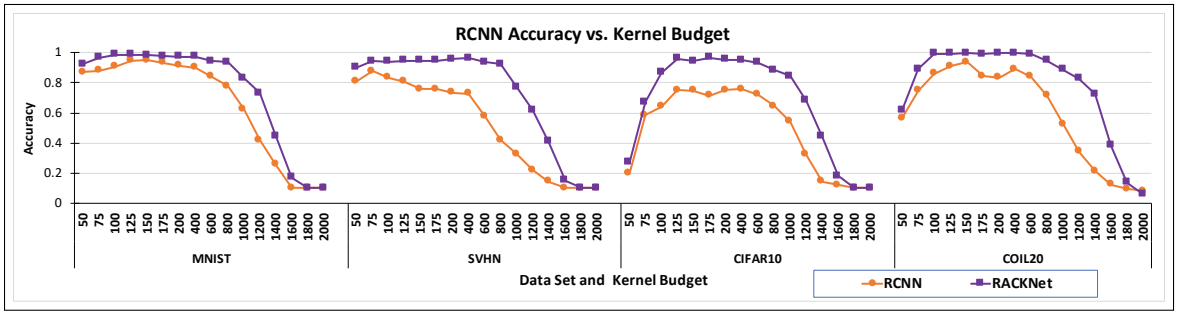
Table 3.8: Execution Time (*in seconds*)

Datasets (# $of\ Instances$)	**MNIST** $(60K)$	**SVHN** $(73K)$	**CIFAR** $(50K)$	**COIL** $(1.3K)$	**ImageNet** $(1.23M)$
Feature	340.34	449.35	302.10	110.59	8439.32
Entropy Histogram	81.04	119.47	60.41	26.56	4110.04
Mixture Separation	1.79	1.71	2.17	1.93	3.76
Clustering	75.76	97.45	62.13	30.21	1529.67
Training	929.33	1396.79	1189.61	187.01	31952.12

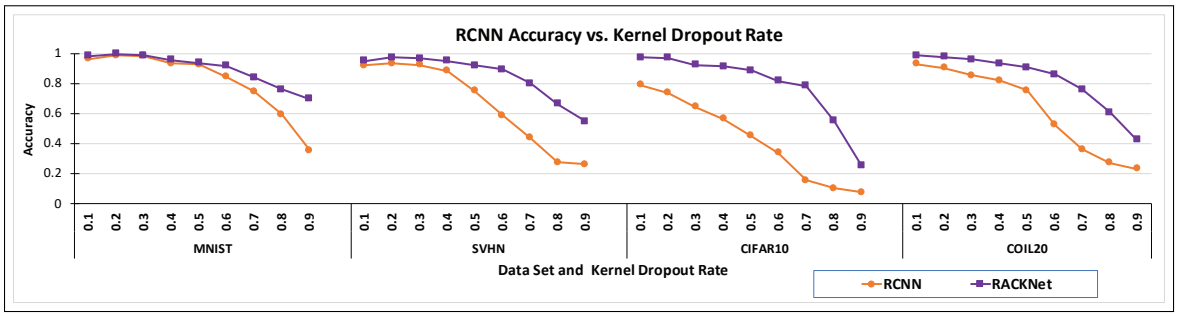
network configurations during the hyper-parameter search, RACKNet trains only a **single** network configuration; instead, it pre-processes the data to determine the best performing configuration Since the pre-processing cost grows much slower that the network training cost, this provides large cost savings.

3.6 Conclusion

In this chapter, the proposed RACKNet, a framework for allocating convolution kernels for different layers of a CNN is presented. In particular, noting that local image features pre-extracted from the training data can provide reliable insights that can inform the design of the CNN architectures, and present four key observations that link sizes, complexities, and distributions of these local features to CNN hyper-parameters. Experiments using several benchmark datasets have shown that the proposed approach leads to highly accurate classifiers without requiring hand-crafting of the hyper-parameters. Experiments have further shown that the proposed framework leads to more-accurate CNNs that are robust against kernel budget availabilities, dropout rates, and Racks.

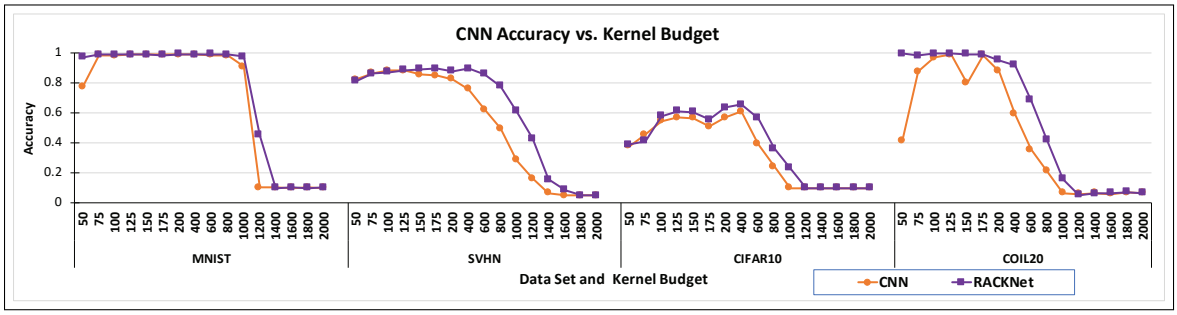


(a) Classification Accuracy vs Kernel Budget

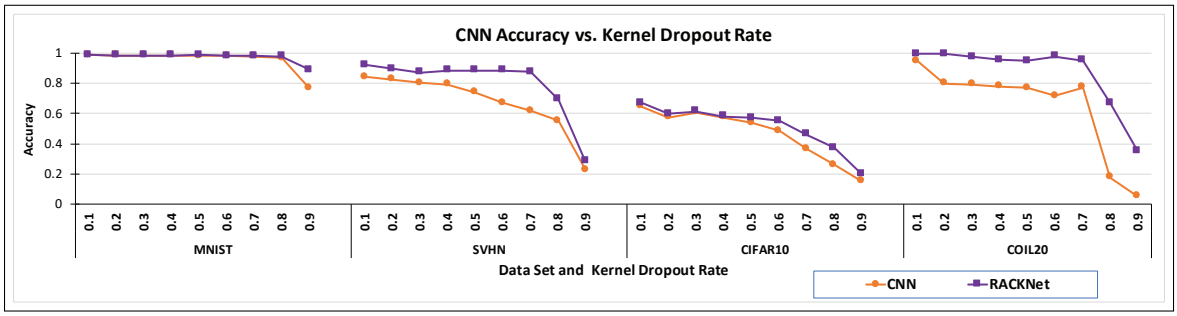


(b) Classification Accuracy vs Kernel Dropout Rate

Figure 3.7: Accuracy vs. Kernel Budget and Dropout Rates (RCNN Implementation), for Clarity, RCNN with Uniform-Layer Budget Allocation Strategy is Shown.



(a) Classification Accuracy vs Kernel Budget



(b) Classification Accuracy vs Kernel Dropout Rate

Figure 3.8: Accuracy vs. Kernel Budget and Dropout Rates (CNN Implementation), for Clarity, CNN with Uniform-Layer Budget Allocation Strategy is Shown.

Chapter 4

iSparse: OUTPUT INFORMED SPARSIFICATION OF NEURAL NETWORK

4.1 Overview

Deep neural networks have demonstrated unprecedented success in various multimedia applications. However, the networks created are often very complex, with large numbers of trainable edges which require extensive computational resources. It must be noted that many successful networks nevertheless often contain large numbers of redundant edges. Moreover, many of these edges may have negligible contributions towards the overall network performance. This chapter presents the novel *iSparse framework*[1], and experimentally show, that one can sparsify the network without impacting the network performance. iSparse leverages a novel edge significance score, $\boldsymbol{E}$, to determine the importance of an edge with respect to the final network output. Furthermore, iSparse can be applied both while training a model or on top of a pre-trained model, making it a retraining-free approach - leading to a minimal computational overhead. Comparisons of iSparse against state-of-the-art techniques on benchmark datasets show that iSparse leads to effective network sparsification for architectures such as, LeNet, VGG and ResNet.

[1] Garg, Yash, and Candan, K. Selçuk. iSparse: Output Informed Sparsification of Neural Network. In Proceedings of the 2020 International Conference on Multimedia Retrieval (pp. 180-188).

4.2 Introduction

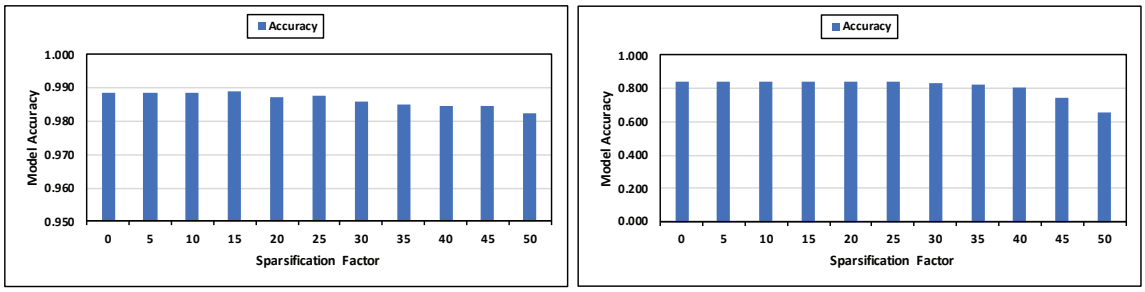
Deep Neural Networks (DNNs), particularly Convolutional Neural Networks (CNNs), have shown impressive success in many applications, such as facial recognition [64], time series analysis [121], speech recognition [47], object classification [70], and video surveillance [57]. As the term *"deep" neural networks implies*, this success often relies on large networks, with large number of trainable edges (weights) [46, 52, 106, 132].

4.2.1 Challenge: Network Size and Complexity

While a large number of trainable edges help generalize the network for complex and diverse patterns in large-scale datasets, this often comes with enormous computation cost to account for the non-linearity of the deep networks ("*relu*", "*sigmoid*", "*tanh*"). In fact, CNNs owe their recent success to hardware level innovations that render the immense computational requirements practical [82, 90]. However, the benefits of hardware solutions and optimizations that can be applied to a general purpose DNN or CNN are limited and these solutions are fast reaching their limits. This has lead to significant interest in network-specific optimization techniques, such as network compression [21], pruning [69, 123], and regularization [108, 115], aim to reduce the number of edges in the network. However, many of these techniques require retraining the pruned network, leading to the significant amount of computational waste.

4.2.2 Network Sparsification

Many successful networks often contain large numbers of redundant edges, for example, the weights of sample network shown in Figure 4.2, highlights that the



(a) MNIST Dataset (b) CIFAR10 Dataset

Figure 4.1: Comparison of Model Classification Accuracy and Model Sparsification Factor for Different Image Classification Datasets

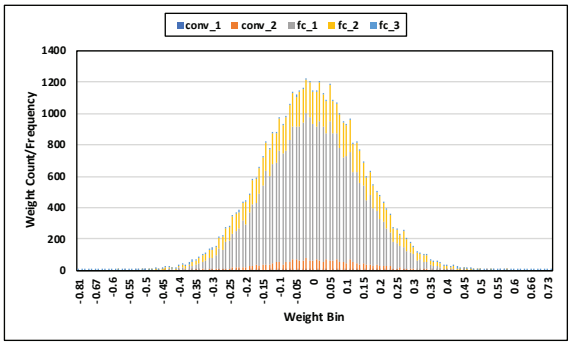


Figure 4.2: Overview of Post-Training Model Parameter Distribution for LeNet-5 Architecture When Trained for MNIST Dataset

weight distribution is centered around zero and has significant number of weights with insignificant contribution to the network output. Such edges may add noise or non-informative information leading to reduction in the network performance. [5, 27, 123] has shown that it is possible to predict 95% network parameters while only learning 5% parameters. Sparsification techniques can generally be classified into neuron/kernel sparsification [69, 123] and edge/weight sparsification techniques [5, 115]: [69] proposed to eliminate neurons that have low $l2$-norms of their weights, whereas [123] proposed a neuron importance score propagation (NISP) technique where neuron importance scores (using [97] - See Equation 4.6) are

propagated from the output layer to the input layer in a back-propagation fashion. Dropout [108] technique instead deactivates neuron activations at random. As an edge sparsification technique, DropConnect [115] selects edges to be sparsified randomly. [5] showed that the network performance can be maintained by eliminating insignificant weights without modifying the network architecture.

4.2.3 iSparse Contributions

Following these works, it can be argued that network sparsification can be a very effective tool for reducing the size and complexity of DNNs without any significant loss in accuracy. However, it can also be argued that the edge weights cannot be used "*as is*" for pruning the network. Instead, one needs to consider the *significance* of each edge within the context of their place in the network (Figure 4.3):

> "*two edges with the same edge weight may have different degrees of contributions to the final network output*"

and this work demonstrates that it is possible to *quantify significance of each edge in the network*, relative to their contributions to the final network output and use these measured edge significance to minimize the redundancy in the network by sparsifying the weights(edges) that contributes insignificantly to network. Through experimental evaluation of *iSparse framework* has shown that a given pre-trained network can be sparsified by almost 50% without impacting the network performance. The *key contributions* of iSparse are as follows:

- **Output-informed quantification of the significance of network parameters**: Informed by the final network layer output, iSparse computes and propagates edge significance scores that measure the importance of each and every edge in the network with respect to the model output (Section 4.4).

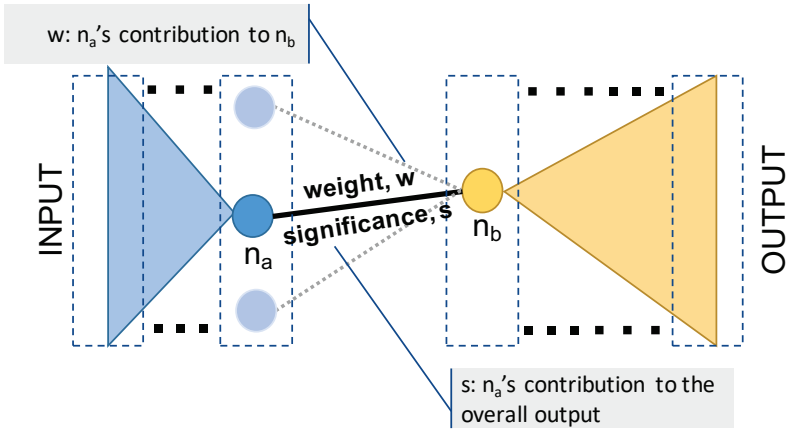


Figure 4.3: Overview of iSparse Framework, Considering the Contribution of Neuron n_i to the Overall Output Rather than Only Between n_i and n_j Neurons

- **Retraining-free network sparsification** (*Sparsify-with*): The proposed iSparse framework is robust to edge sparsification and can maintain network performance without having to retraining the network. This implies that one can apply iSparse on pre-trained networks, on-the-fly, to achieve the desired level of sparsification (Section 4.4.3)

- **Sparsification during training** (*Train-with*): iSparse can also be used as a regularizer while training model, allowing for learning of sparse networks from scratch (Section 4.5).

As the sample results in Figure 4.1 shows, iSparse is able to achieve 30-50% sparsification with minimal impact on model accuracy. More detailed experimental comparisons (See Section 4.6) of iSparse against Dropout, l_1, DropConnect, Retraining-Free and Lottery-Ticket Hypothesis on benchmark datasets illustrated that iSparse leads to more effective network sparsifications.

4.2.4 Organization of the Chapter

The chapter is organized as follows: Section 4.3 provides an overview of the work done in the domain and the crucial background on the state-of-the-art network architectures in use. Section 4.4 describes in detail the proposed iSparse framework (*sparsify-with*) and Section 4.5 provides the iSparse (train-with) framework. Section 4.6 presents the experimental evaluation of the proposed framework and Section 4.7 concludes the chapter.

4.3 Related Work

A *neural network* is a sequence of layers of neurons to help learn (and remember) complex non-linear patterns in a given dataset [39]. Recently, Deep Neural Networks (DNNs), and particularly CNNs, have leveraged hardware optimizations to scale large networks and have shown impressive success in several data analysis and machine learning applications [47, 57, 64, 70, 121].

The number of trainable parameters in a network can range from as low as tens of thousands [67] to hundreds of millions [106]. The three order increase in the number trainable parameter may lead to parameters being redundant or parameters that have negligible contribution to the overall network output. This redundancy and insignificance of the network parameters has led to advancements in network regularization, by introducing dynamic or informed sparsification in the network. These advancements can be broadly classified into two main categories: parameter pruning and parameter regularization. In particular, pruning focuses on compressing the network by eliminating the redundant or insignificant parameters. [43, 42] aims to prune the parameters with near-zero weights inspired from l_1 and l_2 regularization [111, 112]. [69] choose to filter out convolutional kernel with

cumulative minimum weight in given layer. Recently, [123] minimizes the change in final network performance by eliminating the neuron that have minimal impact on the network output by leveraging neuron importance score ($\boldsymbol{N}_L$) - computed using *Inf-FS* [97]. More complex approaches have been proposed to tackle the problem of redundancy in the network through weight quantization. [95] propose to the quantize the inputs and output activations of the layers in a CNN by using step function and also leveraging the binary operation by using the binary weights opposed to the real-values weights. [19] focuses on low-level mobile hardware with limited computational power, and proposed to leverage the network's inherent redundancy by using hashing functions to compress the network weights.

[7, 119] showed that each input feature to a given layer in the network rarely have the same importance, therefore, learning there individual importance (attention) helps improve the performance of the network. More recently, [32] has shown that *input data* informed deep networks can provide high-performance network configurations. Further, works such as [33, 34, 72] have shown that salient information can help improve the model performance. iSparse relies on *output information* for identifying and eliminating insignificant parameters from the network, without having to update the edge weights or retraining the network.

4.4 iSparse Framework

As discussed in Section 4.2, in order to tackle complex inputs, deep neural networks have gone increasingly deeper and wider. This design strategy, however, often results in large numbers of insignificant edges[2] (weights), if not redundant. This section, describes in detail, the proposed *iSparse framework* which quantifies the significance of each individual connection in the network with respect to the over-

[2]An edge is defined as a direct connection (weighted) between two neurons.

all network output to determine the set of edges that can be sparsified to alleviate network redundancy and eliminate insignificant edges. iSparse aims to determine the significance of the edges in the network to make informed sparsification of the network, irrespective of the type of layers, such convolutional (n-dimensional) and fully connected (dense).

4.4.1 Mask Matrix

A typical neural network, $\mathcal{N}$, can be viewed as a sequential arrangement of heterogeneous layers ($\mathcal{L}$), such as convolutional ($\mathcal{C}$) and fully connected ($\mathcal{F}$) layers:

$$\mathcal{N}(\boldsymbol{X}) = \mathcal{L}_L\left(\mathcal{L}_{L-1}\ \ldots\ (\mathcal{L}_2\left(\ \mathcal{L}_1(\boldsymbol{X})\right))\right), \tag{4.1}$$

here, $\boldsymbol{X}$ is the input, L is the total number of layers in the network and $\mathcal{L}\ \epsilon\ \{\mathcal{C}, \mathcal{F}\}$, s.t., any given layer, $\mathcal{L}_l \mid 1 \leq l\ \epsilon\ L$, can be generalized as

$$\mathcal{L}_l(\boldsymbol{Y}_l) = \sigma_l(\boldsymbol{W}_l\boldsymbol{Y}_l + \boldsymbol{B}_l) = \hat{\boldsymbol{Y}}_l, \tag{4.2}$$

where, $\boldsymbol{Y}_l$ is the input to the layer (s.t. $\boldsymbol{Y}_l = \hat{\boldsymbol{Y}}_{l-1}$ and for $l = 1$, $\boldsymbol{Y}_1 = \boldsymbol{X}$) and σ_l, $\boldsymbol{W}_l$, and $\boldsymbol{B}_l$ are the activation function, weight, and bias respectively. Note that, if the l^{th} layer has m_l neurons (i.e., $|\hat{\boldsymbol{Y}}_l| = n_l$) and the $(l-1)^{th}$ layer has n_l neurons (i.e., $|\boldsymbol{Y}_l| = m_l$), then $\hat{\boldsymbol{Y}}_l\ \epsilon\ \mathbb{R}^{m_l \times 1}$, $\boldsymbol{Y}_l\ \epsilon\ \mathbb{R}^{n_l \times 1}$, $\boldsymbol{W}_l\ \epsilon\ \mathbb{R}^{m_l \times n_l}$ and $\boldsymbol{B}\ \epsilon\ \mathbb{R}^{m_l \times 1}$.

Given this formulation, the problem of identifying insignificant edges can be formulated as the problem of generating a sequence of binary **mask matrices**, $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_L$, that collectively represents whether any given edge in the network is sparsified (0) or not (1):

$$\boldsymbol{M}_l = \mathbb{B}^{n_l \times m_l}, \mathbb{B}\ \epsilon\ \{0, 1\}, 1 \leq l \leq L \tag{4.3}$$

4.4.2 Edge Significance Score

Let M_l be a mask matrix as defined in Equation 4.3, and M_l can be expanded as

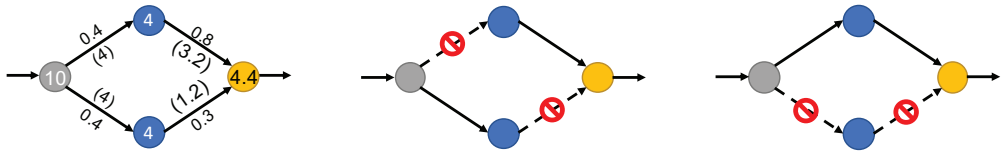
$$M_l = \begin{bmatrix} \mathbf{M}_{l,1,1} & \dots & \mathbf{M}_{l,1,n_l} \\ \vdots & \ddots & \vdots \\ \mathbf{M}_{l,m_l,1} & \dots & \mathbf{M}_{l,m_l,n_l} \end{bmatrix}, \tag{4.4}$$

where each $\mathbf{M}_{l,i,j} \in \{0, 1\}$ corresponds to an edge $e_{l,i,j}$ in the network. The goal of iSparseis to learns an edge significance score to help set the binary value of $\mathbf{M}_{l,i,j}$ for each edge in the network. More specifically, iSparse aims to associate a non-negative real valued number, $\mathbf{E}_{l,i,j} \geq 0$, to each edge in the network, s.t.

$$\mathbf{M}_{l,i,j} = \begin{cases} 1 & \mathbf{E}_{l,i,j} \geq \tau_l(\theta_l) \\ 0 & \mathbf{E}_{l,i,j} < \tau_l(\theta_l) \end{cases}, \forall i = 1 \dots m_l, j = 1 \dots n_l. \tag{4.5}$$

Here, $\tau_l(\theta_l)$ represents the lowest significance of the θ_l% of the most significant edges in the layer l. Intuitively, given a target sparsification rate, θ_l, iSparseranks all the edges based on their edge significance scores and keep only the highest scoring θ_l% of the edges by setting their mask values to 1.

As shown in Figure 4.2, in the Section 4.2, the (signed) weight distribution of the edges in a layer is often centered around zero, with large numbers of edges having weights very close to 0. Also argued in the Introduction that such edges can work counter-intuitively and add noise or non-informative information leading to reduction in the network performance. In fact, several existing works, such as [5], relies on these weights for eliminating insignificant edges without having to retrain the network architecture. However, as commented in the Introduction, iSparse argues that edge weights should not be used alone for sparsifying the network. Instead, one needs to consider each edge within the context of their position in the network:



(a) Sample Network (b) Retrain-Free (c) iSparse

Figure 4.4: A Sample Network Architecture and its Sparsification Using Retraining-Free [5] and iSparse; Here Node Labels Indicate Input to the Node; Edge Labels [0,1] Indicate the Edge Weights; and Edge Labels Between Parentheses Indicate Edge Contribution

two edges in a network with the same edge weight may have different degrees of contributions to the final network output. Unlike existing works, iSparse takes this into account when selecting the edges to be sparsified (Figure 4.4).

More specifically, let $\boldsymbol{W}_l^+$ be the absolute positive of the weight matrix, $\boldsymbol{W}_l$, for edges in l^{th} layer. Next, iSparsecomputes the corresponding edge significance score matrix, $\boldsymbol{E}_l$, as

$$\boldsymbol{E}_l = \boldsymbol{W}_l^+ \odot \boldsymbol{N}_l \tag{4.6}$$

where, $\boldsymbol{N}_l$ represents the neuron significance scores[3], $\mathbf{N}_{l,1}$ through $\mathbf{N}_{l,n_l}$, and " $\odot$ " represents the scalar multiplication between edge weights and neuron scores. This can further be expanded as,

$$\boldsymbol{E}_l = \begin{bmatrix} \mathbf{W}^+_{l,1,1} \times \mathbf{N}_{l,1} & \dots & \mathbf{W}^+_{l,1,n_l} \times \mathbf{N}_{l,1} \\ \vdots & \ddots & \vdots \\ \mathbf{W}^+_{l,m_l,1} \times \mathbf{N}_{l,n_l} & \dots & \mathbf{W}^+_{l,m_l,n_l} \times \mathbf{N}_{l,n_l} \end{bmatrix}. \tag{4.7}$$

$\mathbf{N}_{l,i}$, denotes the significance of the i^{th} input neuron to the l^{th} connection layer of the network, which itself is defined recursively, *based on the following layer in the*

[3] $\boldsymbol{N}_l$ summarizes the edge and neuron importance in the subsequent layers i.e. $\mathcal{L}_{l+1} \dots \mathcal{L}_L$.

network, using the conventional dot product:

$$\boldsymbol{N}_l = \boldsymbol{W}_{l+1}^{+} \boldsymbol{N}_{l+1}, \tag{4.8}$$

that is

$$\boldsymbol{N}_l = \begin{bmatrix} \mathbf{W}_{l+1,1,1}^{+} \times \mathbf{N}_{l+1,1} + \cdots + \mathbf{W}_{l+1,1,n_{l+1}}^{+} \times \mathbf{N}_{l+1,n_{l+1}} \\ \vdots \\ \mathbf{W}_{l+1,m_{l+1},1}^{+} \times \mathbf{N}_{l+1,1} + \cdots + \mathbf{W}_{l+1,m_{l+1},n_{l+1}}^{+} \times \mathbf{N}_{l+1,n_{l+1}} \end{bmatrix}. \tag{4.9}$$

Note that $\boldsymbol{N}_l$ can be expanded as

$$\boldsymbol{N}_l = (\boldsymbol{W}_{l+1}^{+}(\boldsymbol{W}_{l+2}^{+} \ldots (\boldsymbol{W}_{L-1}^{+}(\boldsymbol{W}_{L}^{+} \boldsymbol{N}_L)))), \tag{4.10}$$

Above, $\boldsymbol{N}_L$ denotes the neuron scores of the final output layer, and $\boldsymbol{N}_L$ is defined using *infinite feature selection* [97, 123] (more details can be seen in Section 2.2.2 of Chapter 2) as $\boldsymbol{N}_L = inffs(\hat{\boldsymbol{Y}}_L)$ where $\hat{\boldsymbol{Y}}_L \, \epsilon \, \mathbb{R}^{x \times n}$ (x is the number of input samples and n is the number of output neurons) to determine neuron importance score with respect to the the final network output. Given the above, the edge score (Equation 4.6) can be rewritten as

$$\boldsymbol{E}_l = \boldsymbol{W}_{l}^{+} \odot \left(\boldsymbol{W}_{l+1}^{+}(\boldsymbol{W}_{l+2}^{+} \ldots (\boldsymbol{W}_{L-1}^{+}(\boldsymbol{W}_{L}^{+} \boldsymbol{N}_L)))\right). \tag{4.11}$$

Note that the significance scores of edges in layer l considers not only the weights of the edges, but also the weights of all downstream edges between these edges and the final output layer.

4.4.3 Edge Sparsification

As noted in Section 4.4.1, the binary values in the masking matrix $\boldsymbol{M}_l$ depends on $\tau_l(\theta_l)$, which represents the lowest significance of the θ_l% of the most significant

edges in the layer[4]: therefore, given a target sparsification rate, θ_l, for layer l, iSparse ranks all the edges based on their edge significance scores and keep only the highest scoring θ_l% of the edges by setting their mask values to 1. Note that, once an edge is sparsified, change in its contribution is not propagated back to the layers earlier in the network relative to the sparsified edge. Having determined the insignificant edges with respect to the final layer output, represented in form of the mask matrix, $\boldsymbol{M}_l$ (described in Section 4.4.1), the next step is to integrate this mask matrix in the layer itself. To achieve this, iSparse extends the layer l (Equation 4.2) to account for the corresponding *mask matrix* ($\boldsymbol{M}_l$):

$$\mathcal{L}_l(\boldsymbol{Y}_l) = \sigma_l\left(\left(\boldsymbol{W}_l * \boldsymbol{M}_l\right)\boldsymbol{Y}_l + \boldsymbol{B}_l\right), \tag{4.12}$$

where, $*$ represents the element-wise multiplication between the matrices $\boldsymbol{W}_l$ and $\boldsymbol{M}_l$. Intuitively, $\boldsymbol{M}_l$ facilitates introduction of informed sparsity in the layer by eliminating edges that do not contribute significantly to the final output layer.

4.5 Integration of iSparse within Model Training

Previous section discussed the computation of edge significance scores on a pre-trained network, such as of pre-trained ImageNet models, and the use of these scores for network sparsification. This section further highlights that iSparse can also be integrated directly within the the training process.

To achieve this, the edge significance score is computed for every trainable layer in the network using the strategy described in Section 4.4.2 and the mask matrix is updated using Equation 4.5. Furthermore, the back-propagation accounts for the

[4]In the experiments reported in Section 4.6, without loss of generality, iSparse assumes that θ_l has the same value for all connection layers in the network. This is not a fundamental assumption and iSparse can easily accommodate different rates of sparsification across connection layers.

mask matrices:

$$\boldsymbol{W}_l' = \boldsymbol{W}_l - \eta(\boldsymbol{M}_l * Err_l) \tag{4.13}$$

where, $\boldsymbol{W}_l'$ are the updated weights, $\boldsymbol{W}_l$ original weights, η is the learning rate, and Err_l is the error recorded by as the divergence in between ground truth (Y_l) and model predictions ($\hat{Y_l}$) as $Err_l = |Y_l - \hat{Y_l}|$. Note that, iSparse argues that any edge that does not contribute towards the final model output, must not be included in the back-propagation. Therefore, iSparse masks the error as $Err_l * \boldsymbol{M}_l$.

4.6 Experimental Evaluation

In this section presents the experimental evaluation of the proposed iSparse framework using LeNet, VGG, and ResNet architectures (See Section 4.6.2) and compare it against the approaches, such as Dropout, l_1, DropConnect, Retraining-Free and Lottery-Ticket Hypothesis (see Section 4.6.1). iSparse was implemented in Python environment using Keras Deep Learning Library [22]. All experiments were performed on an Intel Xeon E5-2670 2.3 GHz with 32GB RAM [5].

4.6.1 Competitors

iSparse is compared against several state-of-the-art network regularization and sparsification techniques:

- **Dropout**[6] [108] is a randomly drops neuron activations after every epoch
- **Lasso (l_1) regularization**[6] [111] aims to penalize the large model weights by pushing them towards zero, leading to a simpler network

[5] Results presented in this chapter were obtained using NSF testbed: "*Chameleon: A Large-Scale Re-configurable Experimental Environment for Cloud Research*"

[6] train-with configuration

- **DropConnect** [115] is a purely random approach, where edges are randomly selected for sparsification[6,7]

- **Retraining-free** [5] considers each layer independently and sparsifies insignificant weights in the layer, without accounting for the final network output contribution[6,7]

- **Lottery Ticket Hypothesis** [30] aims at finding a subnetwork configuration by iterative sparsification of network[6]

4.6.2 Benchmark Networks and Datasets

iSparse, without loss of generality, leverages LeNet-5 [67], VGG-16 [106], and ResNet-18 [46] as the baseline architectures to evaluate sparsification performance on different benchmark image classification datasets and for varying degrees of edge sparsification. This section presents the overview of these architectures:

- **LeNet-5**: Designed for recognizing handwritten digits, LeNet-5 is simple network with 5 trainable (2 convolution and 3 dense) and 2 non-trainable layers using average pooling with *tanh* and *softmax* as the hidden and output activation.

- **VGG-16**: VGG [106]'s, a 16 layer network with 13 convolution and 3 dense layers, with interleaved 5 max-pooling layers. VGG leverages ReLU as the hidden activation to overcome the problem of vanishing gradient, as opposed to *tanh*

- **RESNet-18** [46] consists of 17 convolutional layers with varying (64, 128, 256, and 512) number of convolutional kernels, a 2D maxpooling layer, a global

[7] sparsify-with configuration

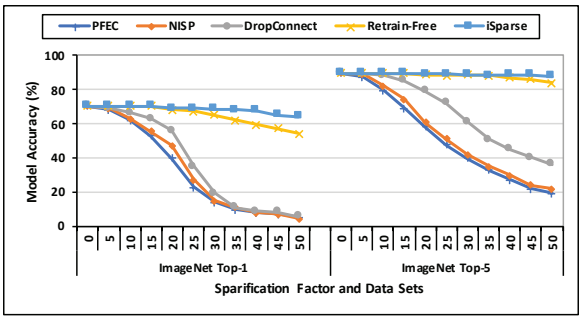


Figure 4.5: Top-1 and Top-5 Classification Accuracy for Sparsified VGG-16 Model for ImageNet (*sparsify-with*)

average layer, and a fully connected layer. First convolutional layer uses a kernel size of 7×7 and the remainder use 3×3 as kernel size

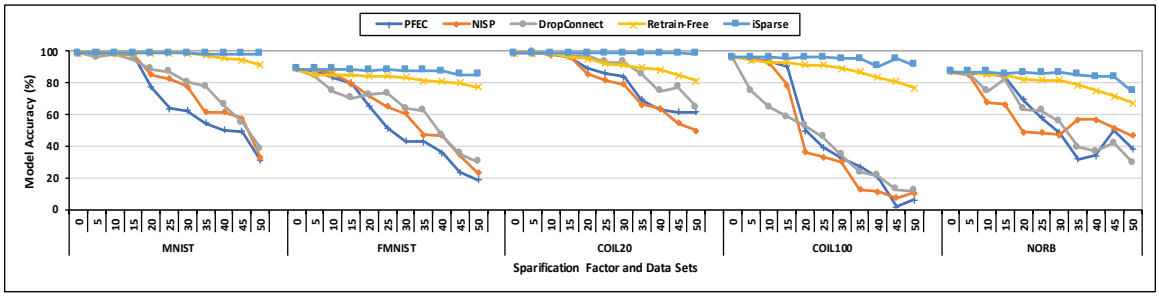
LeNet's simplicity has made it a common benchmark for datasets recorded in constrained environments, such as MNIST [66], FMNIST [120], COIL [88, 87], and NORB [68], and given the ability of VGG/ResNet network to learn the complex pattern in the real-world dataset, benchmark datasets, such as CIFAR10/20/100 [61], SVHN [89], GTSRB [109], and ImageNet [26] are used.

4.6.3 Classification Results

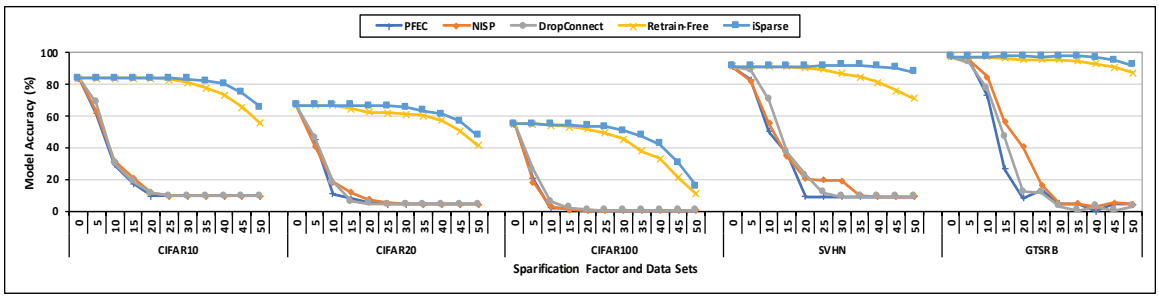
This section presents and discusses the accuracy results.

4.6.3.1 Sparsification of Pre-trained Models (*sparsify-with*)

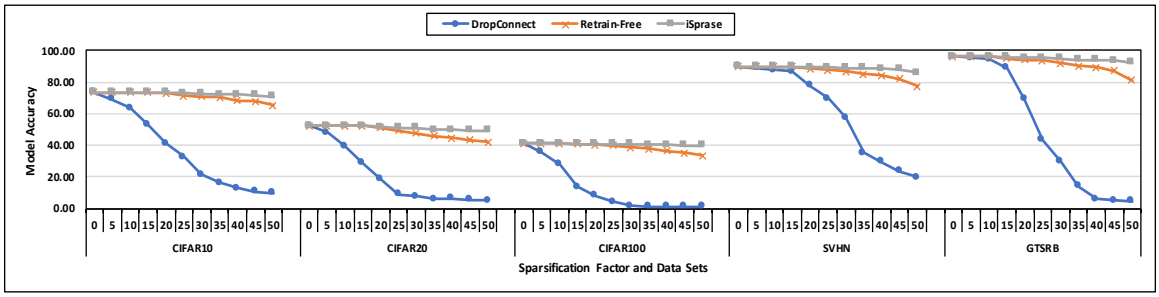
Figure 4.5 presents the top-1 and top-5 classification results for ImageNet dataset for VGG-16 network. As seen in the Figure 4.5, iSparse provides the highest robustness to the degree of sparsification in the network. In particular, with iSparse, the network can be sparsified by 50% with $\leq 6\%$ drop in accuracy for top-1 and $\leq 2\%$ drop in accuracy for top-5 classification, respectively. In contrast, the com-



(a) LeNet Network (LeNet-5)



(b) VGG Network (VGG-16)



(c) ResNet-18 Network

Figure 4.6: Top-1 Classification Accuracy Results for Sparsified Pre-Trained Models (*Sparsify-With*)

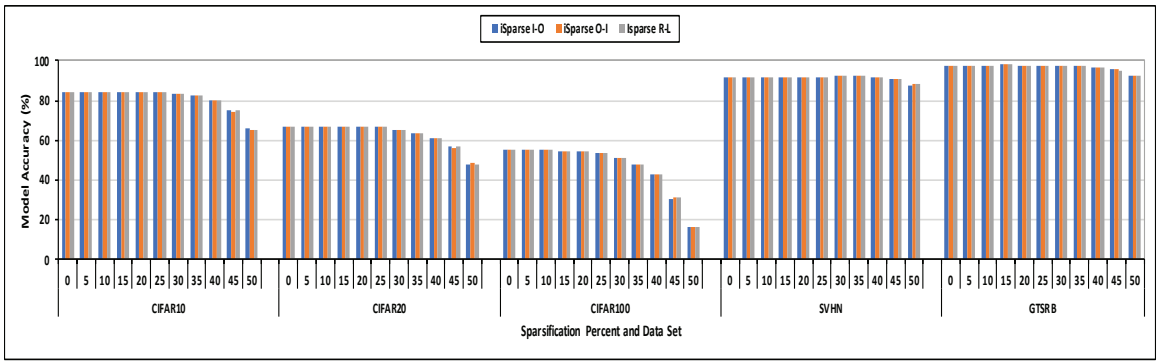


Figure 4.7: Robustness to the Order of the Layer While Sparsifying the Network with iSparse (*Train-With*)

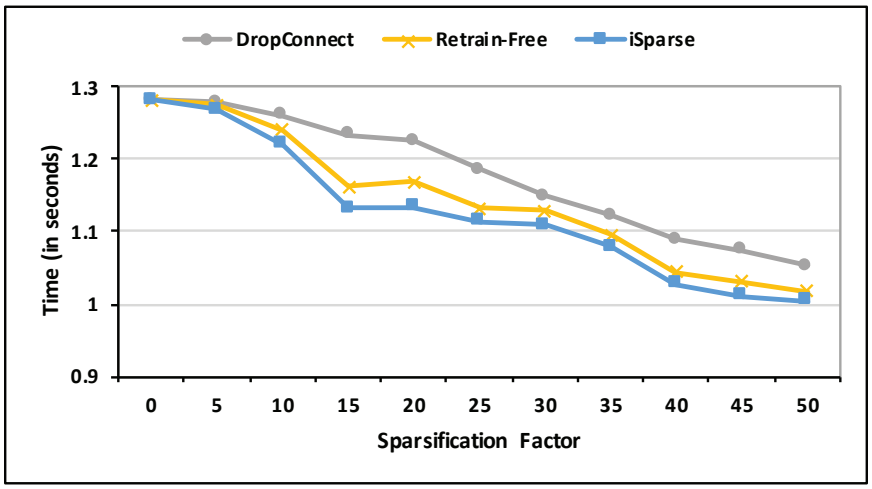


Figure 4.8: Model Classification Time vs Sparsification Factor for 10,000 MNIST Test Images (*Train-With*)

petitors, see larger drops in accuracy. The closest competitor, Retrain-free, suffers a loss in accuracy of $\sim 16\%$ and $\sim 6\%$ for top-1 and top-5 classification, respectively. The other competitors suffer significant accuracy drops after a mere 10-20% sparsification.

Figures 4.6a, 4.6b and 4.6c show the top-1 classification accuracy results for other models and data sets. As shown in the figures, the above pattern holds for all configurations considered: iSparse provides the best robustness. It is interesting

to note that DropConnect sees drastic drop in accuracy for the VGG networks and especially on the CIFAR data. This is likely because, VGG-CIFAR is already relatively sparse ($20\% >$ sparsity as opposed to $\sim 7\%$ for VGG-ImageNet and $< 1\%$ for LeNet) and these three techniques are not able to introduce additional sparseness in a robust manner. A similar pattern is observed in the ResNet architecture, however, DropConnect shows more invariance to random weight drop, it is argued that this is due to the incorporation of residual connections. In contrast, iSparse is able to introduce significant additional sparsification with minimal impact on accuracy.

Figure 4.9 provides the mask matrices created by the different algorithms to visual illustrate the key differences between the competitors. As shown in the figure, DropConnect selects individual edges for sparsification, but only randomly and this prevents the algorithm to provide sufficiently high robustness. Retrain-free and iSparse both select edges in an fine-grained manner: retrain-free uses relies on edge-weights, whereas iSparse complements edge-weight with an edge significance measure that accounts for each edges contribution to the final output within the overall network. As shown in the Figure 4.9 (d) and (e), this results in some differences in the corresponding mask matrices, and these differences are sufficient to provide significant boost in accuracy.

4.6.3.2 Sparsification during Training (*train-with*)

Table 4.1 present accuracy results for the scenarios where iSparse (iS) is used to sparsify the model during the training process. The table also considers Dropout (DO), l_1 regularization, DropConnect (DC), Retrain-Free (RF), and Lottery-Ticket Hypothesis (LT) as alternatives. As shown in table, for all three network architectures, under most sparsification rates, the output informed sparsification approach underlying iSparse leads to highest classification accuracies.

Table 4.1: Top-1 Classification Accuracy for Sparsified Architectures for Different Datasets (*Train-with*). Notations in Section 4.6.3.2. For Baseline (Base) Architecture and L1, the Achieved Sparsification Is Presented in "()"

Networks		LeNet-5 network					VGG-16 network					ResNet-18 network				
Datasets		MNIST data set					CIFAR-10 data set					CIFAR-10 data set				
Base arch.		98.79 (0% sparsification)					84.14 (0% sparsification)					73.61 (0% sparsification)				
Base w. L1		98.22 (21.62% sparsification)					84.51 (13.26% sparsification)					73.98 (27.53% sparsification)				
Alternatives		DO	DC	RT	LT	iS	DO	DC	RT	LT	iS	DO	DC	RT	LT	iS
Spars. Target	5%	98.09	**98.90**	97.99	98.03	98.09	**85.13**	84.21	84.56	84.29	84.66	73.83	73.72	**74.35**	74.10	74.34
	10%	98.07	98.17	98.45	**98.82**	98.56	**84.77**	84.35	84.33	84.23	84.35	74.52	73.22	74.49	74.31	**74.78**
	20%	98.70	98.79	98.81	98.60	**98.98**	77.81	82.83	85.04	84.68	**85.81**	**75.42**	69.84	73.08	73.38	75.01
	30%	98.54	98.78	98.71	98.72	**98.98**	76.16	83.41	84.17	83.48	**84.48**	**74.69**	72.43	74.10	73.39	74.36
	40%	98.11	98.85	98.97	98.37	**98.99**	77.51	**83.96**	82.01	81.37	82.21	67.89	70.38	73.15	74.10	**75.27**
	50%	97.60	98.55	98.85	98.68	**99.00**	69.21	83.44	84.15	81.59	**84.53**	66.20	71.35	72.69	**74.13**	74.12
Datasets		FMNIST					CIFAR20					CIFAR20				
Base arch.		88.02 (0% sparsification)					66.96 (0% sparsification)					52.35 (0% sparsification)				
Base w. L1		86.80 (21.84% sparsification)					67.12 (7.57% sparsification)					52.63 (22.94% sparsification)				
Alternatives		DO	DC	RT	LT	iS	DO	DC	RT	LT	iS	DO	DC	RT	LT	iS
Spars. Target	5%	88.35	88.23	88.70	87.76	**88.75**	62.47	62.26	62.01	62.54	**62.85**	52.99	52.42	50.69	51.17	**53.69**
	10%	**88.06**	87.60	87.02	87.34	87.89	66.72	63.63	66.53	66.97	**67.33**	56.16	50.49	51.38	51.44	**57.19**
	20%	87.00	88.02	88.00	86.89	**88.11**	64.10	61.79	63.45	63.89	**65.03**	**56.52**	52.91	54.13	55.60	56.23
	30%	86.51	88.00	87.75	87.37	**88.25**	61.32	62.14	58.39	58.99	**62.14**	52.12	52.97	53.07	54.17	**54.51**
	40%	84.09	87.81	87.25	86.79	**88.30**	65.67	66.34	65.09	65.28	**66.62**	47.17	50.19	51.06	48.26	**51.64**
	50%	83.13	87.76	88.15	86.83	**88.46**	66.77	65.37	65.04	65.56	**68.82**	43.29	46.32	49.35	50.77	**50.98**
Datasets		COIL20					CIFAR100					CIFAR100				
Base arch.		95.96 (0% sparsification)					55.09 (0% sparsification)					41.01 (0% sparsification)				
Base w. L1		95.61 (32.46% sparsification)					55.31 (6.71% sparsification)					45.01 (22.71% sparsification)				
Alternatives		DO	DC	RT	LT	iS	DO	DC	RT	LT	iS	DO	DC	RT	LT	iS
Spars. Target	5%	95.33	95.00	94.16	**95.99**	94.72	53.90	**57.35**	52.74	53.17	53.10	**43.84**	40.36	39.53	40.68	40.71
	10%	95.44	94.72	95.01	95.49	**95.55**	51.24	51.32	51.01	**51.60**	51.48	**42.37**	40.88	38.08	40.77	40.89
	20%	94.33	95.83	95.95	95.01	**96.10**	49.21	**51.67**	50.01	51.11	50.52	42.22	40.90	43.84	43.85	**44.95**
	30%	94.32	95.00	94.74	94.56	**95.00**	53.76	51.19	54.05	54.26	**54.50**	41.58	41.22	43.39	44.99	**45.98**
	40%	92.83	95.00	95.31	94.85	**95.56**	53.91	50.19	55.11	55.26	**55.68**	37.74	41.08	43.25	43.33	**44.51**
	50%	92.61	95.00	94.16	93.26	**94.72**	52.01	51.65	51.78	51.96	**52.56**	39.22	36.57	43.69	43.40	**44.21**
Datasets		COIL100					SVHN					SVHN				
Base arch.		90.00 (0% sparsification)					91.33 (0% sparsification)					89.68 (0% sparsification)				
Base w. L1		90.38 (21.57% sparsification)					91.86 (8.63% sparsification)					88.24 (23.03% sparsification)				
Alternatives		DO	DC	RT	LT	iS	DO	DC	RT	LT	iS	DO	DC	RT	LT	iS
Spars. Target	5%	90.27	89.83	90.12	**91.00**	90.61	92.02	**93.49**	92.42	91.90	92.70	**92.77**	90.79	90.99	90.64	91.98
	10%	89.94	90.05	89.81	89.99	**90.33**	91.99	**95.00**	92.56	92.06	92.96	**91.52**	91.87	89.87	90.06	91.12
	20%	89.32	90.83	88.54	90.56	**91.03**	87.13	92.63	93.34	93.51	**93.74**	90.23	89.83	88.48	89.85	**91.56**
	30%	89.13	90.12	90.12	89.86	**90.55**	89.94	91.95	91.97	82.04	**92.26**	91.68	91.83	87.91	88.98	**92.48**
	40%	86.27	88.22	90.01	90.03	**90.44**	88.65	88.79	91.92	92.12	**92.31**	90.18	91.16	89.83	90.87	**91.87**
	50%	82.77	90.66	90.21	89.99	**90.85**	**88.75**	82.94	87.61	87.75	87.68	89.58	90.79	87.49	88.63	**91.56**
Datasets		NORB					GTSRB					GTSRB				
Base arch.		84.42 (0% sparsification)					97.68 (0% sparsification)					96.12 (0% sparsification)				
Base w. L1		83.30 (27.98% sparsification)					97.74 (35.88% sparsification)					92.02 (44.51% sparsification)				
Alternatives		DO	DC	RT	LT	iS	DO	DC	RT	LT	iS	DO	DC	RT	LT	iS
Spars. Target	5%	**86.46**	84.51	84.75	85.14	85.03	97.11	96.49	97.68	97.98	**98.64**	96.71	95.84	97.19	96.35	**97.23**
	10%	83.25	85.35	85.98	85.75	**86.25**	92.73	**95.03**	91.09	87.32	93.24	97.32	95.71	95.84	96.04	**97.56**
	20%	**85.65**	83.04	82.56	82.98	83.29	94.82	93.02	94.41	90.56	**95.76**	**97.55**	93.26	95.30	95.91	96.96
	30%	85.02	**86.69**	85.91	85.79	86.08	96.30	97.18	97.43	87.27	**98.56**	96.37	94.56	92.70	92.88	**98.81**
	40%	84.72	83.14	84.12	84.86	**85.08**	89.03	95.61	96.91	87.78	**97.61**	**96.25**	93.39	92.90	93.61	94.55
	50%	84.63	84.49	85.61	85.69	**86.77**	84.34	94.31	95.91	89.44	**97.31**	**97.73**	95.91	94.30	93.26	95.12

Table 4.2: Robustness Analysis for Cifar10 (VGG) for Edge-based Sparsification Strategies Against iSparse (*Train-with*)

		Tanh-Adam			Tanh-RMS			ReLU-Adam		
Base arch.		87.66 (0% sparsified)			81.95 (0% sparsified)			84.14 (0% sparsified)		
Alternatives		**DC**	**RT**	**iS**	**DC**	**RT**	**iS**	**DC**	**RT**	**iS**
Sparsification Target	**5%**	88.87	89.56	**89.66**	81.17	85.94	**86.14**	84.21	84.56	**84.66**
	10%	84.75	88.21	**88.31**	86.75	87.01	**87.42**	84.35	84.33	**84.35**
	15%	81.78	81.89	**82.49**	88.75	88.99	**89.04**	83.43	83.51	**83.69**
	20%	85.64	85.95	**86.21**	87.10	89.10	**89.23**	82.83	85.04	**85.81**
	25%	86.10	87.68	**87.86**	84.81	87.31	**87.74**	84.07	84.32	**84.56**
	30%	**87.07**	86.21	86.30	80.20	83.45	**83.56**	83.41	84.17	**84.48**
	35%	82.38	86.43	**86.48**	85.91	86.75	**86.93**	83.06	84.95	**85.42**
	40%	86.76	88.63	**88.77**	82.17	84.10	**87.44**	**83.96**	82.01	82.21
	45%	83.90	85.52	**85.61**	85.79	86.40	**89.42**	82.75	82.81	**82.83**
	50%	**80.45**	79.21	79.59	83.89	86.42	**86.52**	83.44	84.15	**84.53**

4.6.4 Robustness to the Variations in Network Elements

This section presents the impact of various network elements on network performance. In particular, the performance of iSparse (iS) against DropConnect (DC) and Retrain-Free (RF) for different hidden activation functions and network optimizers are compared. Tables 4.2 presents classification performances for networks that rely on different activation functions (*tanh* and ReLU) and for optimizers (Adam and RMSProp). As shown in these two tables, iSparse remains the alternative which provides the best classification accuracy under different activation/optimization configurations.

4.6.5 Ablation Studies for Final Layer Neuron Score Initialization

This section presents the evaluation of the performance on iSparse framework with different type of neuron score initialization strategies for the final layer (L). In particular, the following scoring strategies are evaluated:

– Identity score (IDN) (all scores are "1"),

– Principal Component Analysis (PCA) [92], and

– Infinite Feature Selection (InfFS) [97].

It is observed that the iSparse framework with InfFS as a mechanism to determine the neuron scores for the final layers leads to a better performance opposed to the other strategies (see Table 4.3), as a result of leveraging the relationship between a neuron and other neuron in the layer. A networks capability is not only in predicting the correct label, but also sufficient separating the predicted labels from other possible labels, whereas IDN assumes a total independence among the neurons.

4.6.6 Robustness to the Variations in Sparsification Order

Next, the performance of iSparse under different orders in which the network layers are sparsified is evaluated. In particular, the following *three* sparsification orders are considered:

– *input-to-output layer order*: this is the most intuitive approach as it does not require edge significance scores to be revised based on sparsified edges in layers closer to the input;

– *output-to-input layer-order*: in this case, edges in layers closer to the network output are sparsified first – but, this implies that edge significance scores are

Table 4.3: Model Accuracies for Different Strategies to Initialize Neuron Importance for Final Layer - CIFAR10

Networks		**VGG-16**			**ResNet-18**		
Base arch.		84.14 (0% sparsified)			73.61 (0% sparsified)		
Alternatives		**IDN**	**PCA**	**InfFS**	**IDN**	**PCA**	**InfFS**
Sparsification Target	**5%**	82.94	83.96	**84.66**	73.25	73.20	**74.13**
	10%	81.59	83.85	**84.35**	73.46	73.86	**74.28**
	15%	81.24	83.45	**83.69**	73.32	73.57	**74.12**
	20%	82.67	84.65	**85.81**	74.52	74.21	**75.01**
	25%	81.42	83.98	**84.56**	74.86	74.45	**74.95**
	30%	80.64	82.96	**84.48**	73.72	73.80	**74.36**
	35%	80.45	81.56	**85.42**	73.53	73.99	**74.01**
	40%	78.48	79.54	**82.21**	74.94	75.09	**75.27**
	45%	80.23	81.24	**82.83**	73.06	72.39	**73.39**
	50%	80.85	82.68	**84.53**	74.19	74.19	**74.21**

updated in the earlier layers in the network to account for changes in the overall edge contributions to the network;

– *random layer order*: in this case, to order of the layers to be sparsified is selected randomly.

Figure 4.7 presents the sparsification results for different orders, data sets, and sparsification rates. As evident in the figure, the performance of iSparse is not sensitive to the sparsification order of the network layers.

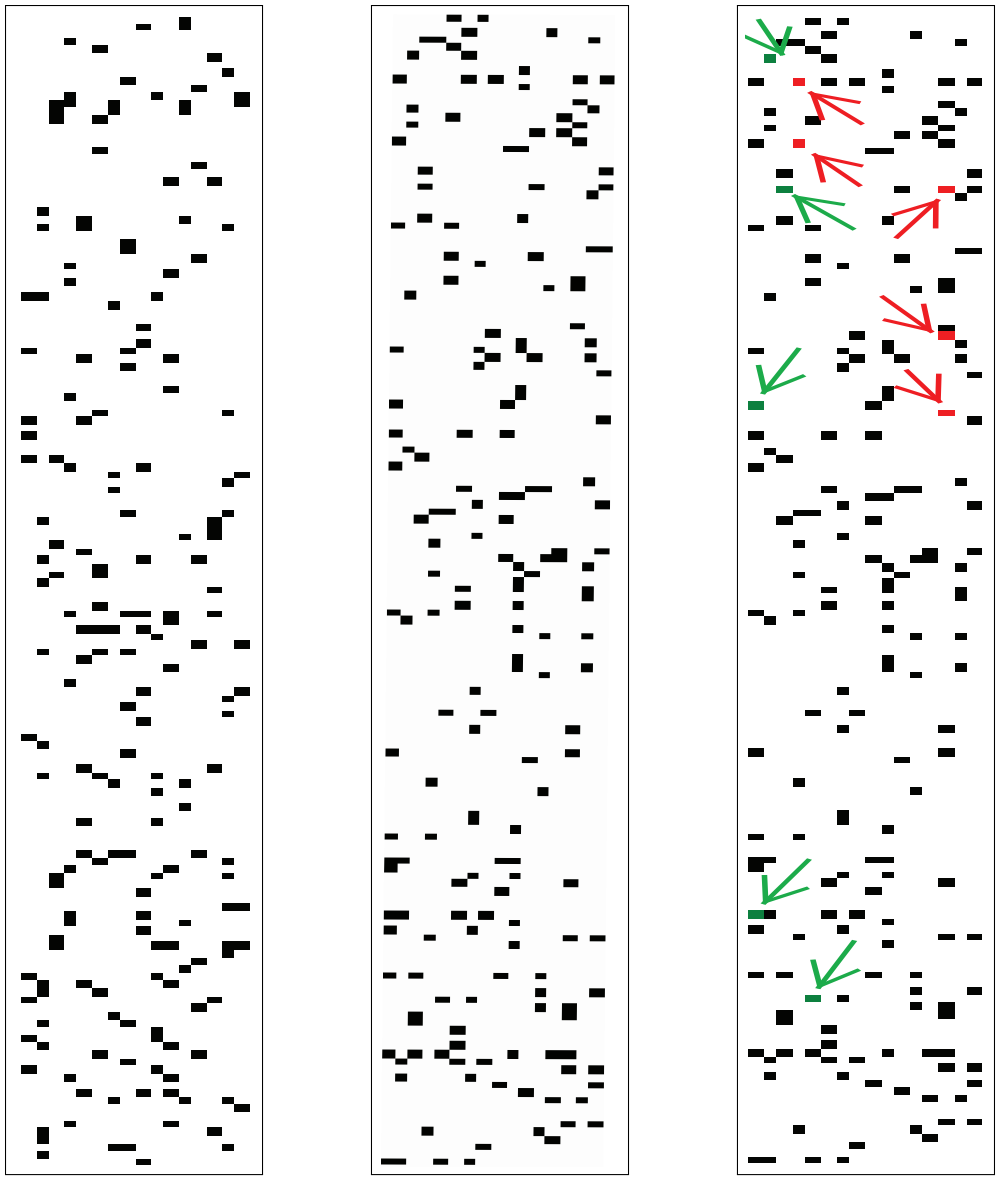
4.6.7 *Impact of Sparsification on Classification Time*

Finally, in Figure 4.8, the impact of edge sparsification on the classification time is investigated. As shown in this Figure, edge sparsification rate has direct impact on the classification time of the resulting model. When it is considered that iSparse allows for $\sim 30 - 50\%$ edge sparsification without any major impact on classification accuracies, this indicates that iSparse has the potential to provide significant performance gains. What is especially interesting to note in Figure 4.8 is that, while all three sparsification methods, iSparse (iS), DropConnect (DC), and Retrain-Free (RF), all have the same number of sparsified edges for a given sparsification factor, the proposed iSparse approach leads to the least execution times among the three alternatives. It can be argued that this is because the output informed sparsification provided by iSparse allows for more efficient computations in the sparsified space.

4.7 Conclusions

This chapter presents the proposed iSparse, a novel output-informed, framework for edge sparsification in deep neural networks (DNNs). In particular, iSparse proposes a novel edge significance score that quantifies the significance of each edge in the network relative to its contribution to the final network output. iSparse leverages this edge significance score to minimize the redundancy in the network by sparsifying those edges that contribute least to the final network output. Experiments, with 11 benchmark datasets and using two well-know network architectures have shown that the proposed iSparse framework enables $30 - 50\%$ network sparsification with minimal impact on the model classification accuracy. Experiments have also shown that the iSparse is highly robust to variations in net-

work elements (activation and model optimization functions) and that iSparse provides a much better accuracy/classification-time trade-off against competitors.



(a) DropConnect [115] (b) Retrain-Free [5] (c) iSparse

Figure 4.9: Mask Matrices for the Lenet Network Conv_2 Layer for Mnist Data (Sparsification Factor = 50%): Dark Regions Indicate Sparsified Edges; In (E) iSparse, the Arrows Point to Those Edges That Are Subject to Different Pruning Decision from Retrain-free in(D) (Green Arrows Point to Edges That Are Kept in iSparse instead of Being Pruned and Red Arrows Point to Edges That Are Sparsified in iSparse instead of Being Kept)

Chapter 5

SAN: SCALE-SPACE ATTENTION NETWORK

5.1 Overview

Deep Neural Networks (DNNs), especially Convolutional Neural Networks (CNNs), have been effective in various data-driven applications. Yet, DNNs suffer from several major challenges; in particular, in many applications where the input data is relatively sparse, DNNs face the problems of overfitting to the input data and poor generalizability. This brings up several critical questions: "Are all inputs equally important?" "Can one selectively focus on parts of the input data in a way that reduces overfitting to irrelevant observations?" Recently, attention networks showed success in helping the overall process focus onto parts of the data with higher importance in the current context. Yet, SAN notes that the current attention network design approaches are not sufficiently informed about the key data characteristics in identifying salient regions in the data. This chapter presents an innovative robust feature learning framework, *scale-invariant attention networks* (SAN)[1], that identifies salient regions in the input data for the CNN to focus on. SAN concentrates attention on parts of the data where there is major change across space and scale, and argue that the salient regions identified by SAN lead to better network performance compared to state-of-the-art (attentioned and non-attentioned) approaches, including state-of-the-art architectures on common benchmark datasets for image and time classification, and object detection.

[1]Garg, Yash, Candan, K. Selçuk, and Sapino, Maria-Luisa. SAN: Scale-Space Attention Networks. In 2020 IEEE 36th International Conference on Data Engineering (pp. 853-864).

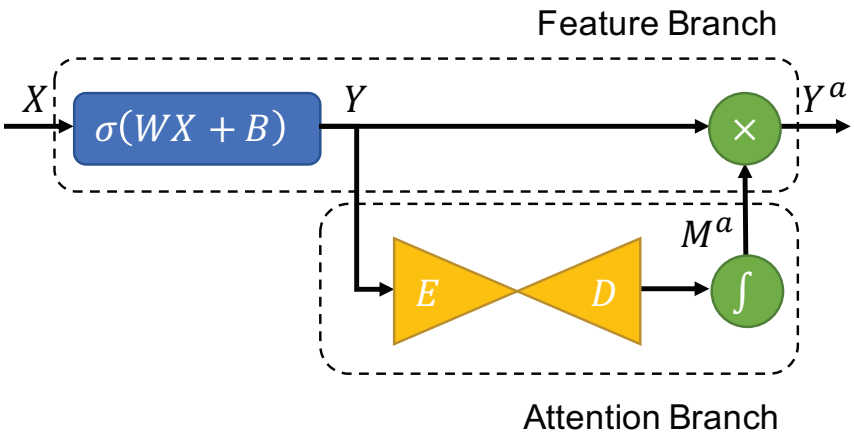


Figure 5.1: Overview of Conventional Attention Module

5.2 Introduction

Deep Neural Networks (DNNs), including Convolutional Neural Networks (CNNs) have seen successful applications in many data engineering domains, such as text processing [100, 78], data alignment [24], recommender systems [18], time series search and processing [122, 130, 121], and media search and analysis [64, 47, 70, 102, 101]. More recently, CNNs' successful application in a variety of data-intensive domains has led to a shift away from feature-driven algorithms into the design of CNN architectures crafted for specific datasets and applications.

CNNs owe their success to large *depth* and *width*: this introduces a large number of trainable parameters (from tens of thousands [66] to hundreds of millions [106]) and enables learning of a rich and discriminating representation of the data [46, 66, 106, 110, 116]. However, as [44] points out, "*increase in the depth of the network can lead to model saturation or even degradation in accuracy*". More specifically, in many applications where the input data is relatively sparse, DNNs face the problems of overfitting to the input data and poor generalizability. This brings up several

critical questions: “Are all inputs equally important?” “Can one selectively focus on parts of the input data in a way that reduces overfitting to irrelevant observations?” Works such as [72, 34, 32] has shown that salient information can help improve the model performance, both deep learning and machine learning models.

5.2.1 Attention Networks

The need for working with a limited number of trainable parameters to learn high performing network architectures necessitates techniques to help focus on the most relevant parts of the data. One way to achieve this is through fusion of multi-modal data characteristics, such as channel (i.e. latent) and spatial relationships in images and temporal relationship in time series, where information transferred from different modalities help strengthen and weaken their individual impacts [116]. Another common approach is to learn multi-scale features [110] to capture a rich representation of data. More recently, attention networks gained popularity as a more effective way to tackle this challenge [7]. Commonly, networks with attention modules have two, *feature* and *attention*, branches (see Figure 5.1). The feature branch is analogous to the conventional networks where the neural structure extracts features from data, whereas, in the attention branch, the network aims to quantify the importance of the input features to focus on. The attention mechanism, on the other hand, enables the network to focus on a specific, contextually-relevant, subset of the features [83].

Attention mechanisms have been developed for different types of data. For instance, the original attention network (proposed by Bahdanau in [7]) was designed to recover attention to help identify a subset of input features important to a given state in a recurrent neural network (RNN) used for analyzing sequence data. In image analysis, on the other hand, the attention branch may aim to translate the

spatial and channel level contextual relationship into an attention mask [116]. Since the introduction of the attention mechanism, there has been significant amount of work done that has enabled state-of-the-art networks, enriched with attention mechanisms, to outperform their predecessors [83, 116, 119, 91, 114]. While these and other works, some of which discussed in Section 5.3, have provided strong evidence regarding the promise of the attention mechanisms in reducing overfit and improving accuracy, SAN notes that the current approaches suffer from a shared shortcoming: *While the existing mechanisms leverage multi-modal information, they fail to consider information that a cross-scale examination of the latent features may reveal.*

5.2.2 Contributions: Scale-Space Attention Networks (SAN)

In a recent work, [32] has shown that scale-space based approaches can be used to inform the design of CNNs – in particular, even though localized features, like SIFT, may not lead to very accurate classifiers themselves, the information these features capture at different scales might nevertheless be used to inform the design of the hyper-parameters (e.g. number of layers, number of kernels per layer) of CNNs. SAN is build on a similar observation and argue that a *scale-space* driven technique can also be used to design better attention mechanisms that can help *focus* attention of the deep neural network to parts of the data that are most critical.

Fundamentally, a CNN architecture extracts increasingly complex (multi-scale) features through layers of interleaved convolutional and pooling layers, coupled with non-linearity enablers, such as ReLU and tanh functions. this chapter also demonstrated that SAN can adapt to the CNN architecture to implement a robust scale-space based attention mechanism that focuses processing onto contextually salient aspects of the data. In particular, this chapter presents a novel scale-space

attention network, *SAN*, which brings together the following key ideas:

- **Identifying salient changes in scale-space**: Traditional attention mechanism consider layers in isolation when generating attention. SAN further argues that comparing and contrasting latent features from two adjacent layers, to locate salient changes in scale-space, is a more effective attention strategy.

- **Attention to extrema**: Given that a neighborhood in the input is likely to have changes in varying amplitudes, and argue that the attention should be given to extrema, where the changes in scale-space are local maxima.

- **Attention region extraction through smoothed extrema**: SAN translates these extrema into attention masks by applying a convolutional operation around the extrema – this helps avoid noisy artifacts in the latent representation which could adversely affect performance.

As SAN experimentally validate in Section 5.5, the proposed *SAN framework* has the following advantages: (a) SAN detects and describes salient changes in the latent features to identify detailed and diverse attention masks that help boost network performance while retaining finer details of the patterns. (b) SAN consistently performs better than the competitors in both bottleneck and full attention scenarios (see details in Section 5.5.3). (c) SAN framework is able to learn a high-performing network architecture with minimal computational overhead.

5.2.3 Organization of the Chapter

The following sections are organized as follow: Section 5.3 discusses current state-of-the-art approaches and their shortcomings, Section 5.4 presents the pro-

posed framework, in Section 5.5 SAN is evaluted, and in Section 5.6 concludes the chapter.

5.3 Related Work

Successful application of DNNs in diverse domains [64, 130, 121, 47, 70, 102, 101, 57, 105] has motivated the community to devise novel network architectures that outperform the prior art.

5.3.1 Design of DNNs

A common approach to design DNNs is to hand-craft specialized network architecture for specific domain and data. As early as 1998, Lecun [67], proposed a *five* layer convolutional network to detect hand-written digits. The increasing prevalence of more complex datasets, such as ImageNet [26], led to more complex design of the hand-crafted networks [106, 46, 110, 51, 52]. These span from 10s to 100s of layers with hundreds of millions of trainable parameters. While these networks have different architectures, they often leverage common design optimizations that have been shown to improve the network performance. For instance, batch-normalization [55] is used to address the problem of co-variate shift in the network by normalizing individual batch output of the layers to facilitate early convergence of the network; ReLU [86] is used to handle the problem of vanishing gradients by eliminating the negative gradient in the feed forward phase of the network. To help with the design of new architectures or for improving existing ones, recently several hyper-parameter search strategies have been proposed: these include, grid-search [63] and random search [10]. Both strategies perform principled hyper-parameter search and have shown to determine an optimal hyper-parameter configuration, however, they heavily rely on domain expert input to hand-craft hyper-

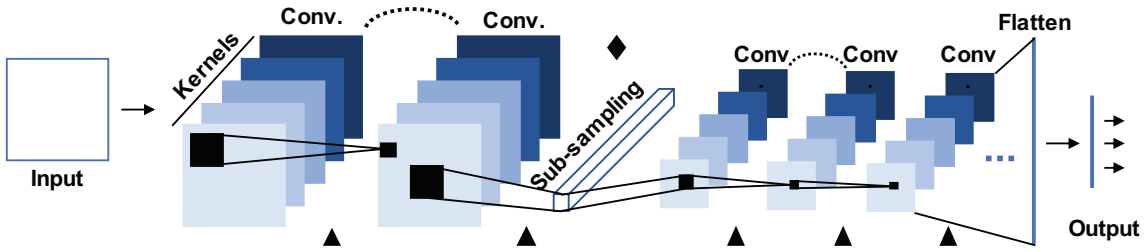


Figure 5.2: Outline of a convolutional neural network [32]: a sequential arrangement of layers with localized spatial connections interleaved with pooling operations that scale the features extracted from the image. In this work, SAN consider two positions for integrating the attention branches: in bottleneck (♭) attention, attention modules is attached right before subsampling (marked with ♦ in the figure); in full (♯) attention, the attention modules are applied at each and every trainable layer (marked with ▲ in the figure)

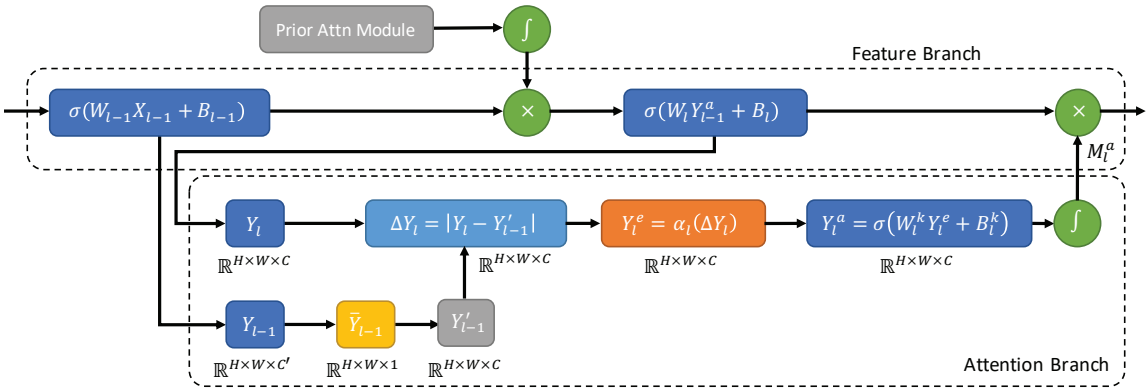
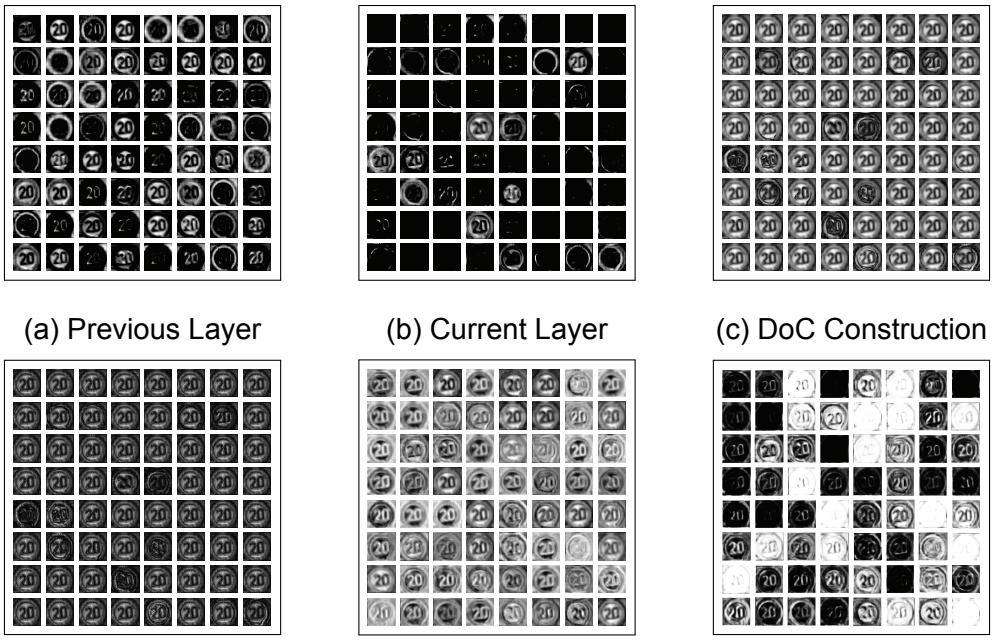


Figure 5.3: Abstract Overview of the Proposed Attention Module

parameter search space. In a recent work, [32] has shown that scale-space based approaches can be used to inform the design of the hyper-parameters (e.g. number of layers, number of kernels per layer) of CNNs.

5.3.2 Attention Networks

Despite these advances, traditional DNNs are still faced with the problem of performance degradation with the increase in the depth [44]: these networks tend to saturate after a certain depth and networks suffer from limited generalization of input data into a fixed-length encoding [7]. Attention mechanisms [7, 116, 119, 91] aim to address this issue. In [7], attention is used for improving the sequence translation task, from *English* to *German*, using recurrent neural nets (RNNs). This work highlighted that not every input feature (word) in a sequence is equally important, rather focusing on a different subset of input features may be more appropriate at different stages of the translation process. Building on this observation, attention has been applied to different applications (image captioning [100], recommender systems [18], multi-task learning [122], question generation [78]) and different network architectures, including CNNs [65] and LSTMs [48]. [116] was one of the early efforts in attentioned image understanding; the authors proposed a residual attention mechanism which emulates residual learning by introducing the attention module as a residual connection comprising of an autoencoder module. Convolutional block attention module [119] and bottleneck attention module [91] proposed to leverage spatial and channel relationships in an image dataset to learn attention masks that summarize the importance of channels in the images and locate where the most information resides spatially. This chapter argues that these works suffer from global (rather than local) summarization and from the fact that they do not leverage cross-layer information for discovering attention. To address this shortcoming, an informed attention network is proposed that leverages salient changes in latent features and transform them into rich (diverse and detailed) attention masks.



(a) Previous Layer (b) Current Layer (c) DoC Construction

(d) Extrema Extraction (e) Extrema Smoothing (f) Learned SAN Mask

Figure 5.4: Sample Outputs of the Various Components of SAN– These Samples Are Taken at the *First* Bottleneck Position in VGG-16, Implementing Attention on the Outputs From *conv_1* ($\boldsymbol{Y}_{l-1}$) and *conv_2* ($\boldsymbol{Y}_{l}$), with 64 Channels (Kernels) Represented here Using an 8×8 Grid. (5.4a) Shows the Output from *conv_1*; (5.4b) Shows the Output Observed at *conv_2*; (5.4c) Shows the DoC Extracted from these two Layers; (5.4d) Highlights the Detected Extrema; and (5.4e) Shows the Output of the Extrema Smoothing Step; Finally, (5.4f) Shows the 64 *Detailed and Diverse* Attention Masks Learned by the Proposed SAN Module

5.4 SAN Framework

As discussed in Section 5.2, the success of deep neural networks can be credited to the increase in their *depth*s and *width*s thanks to modern hardware. This increase has enabled these networks to learn sufficiently complex patterns contained in the dataset. Convolution Neural Networks (CNNs), which seek multi-scale fea-

tures, have proven especially successful in image and time series understanding. However, not every part of the data is of equal importance for extracting features and avoiding overfitting, especially in the presence of sparse data, necessitates the network to learn the importance (*attention*) of different parts of the data. This section presents a novel *scale-space attention network* (*SAN*) framework that identifies salient changes in latent features across scales and translates them into robust (detailed and diverse) attention (Figure 5.3).

5.4.1 Convolutional Neural Networks and Attention Modules

A convolutional neural network (CNN [65]) is a type of neural network that works by leveraging the local spatial arrangements by establishing connections among small spatial regions across adjacent layers (Figure 5.2).

5.4.1.1 Convolutional Neural Architectures

A CNN consists of several complementary components organized into layers:

- Each *convolution* layer links local-spatial data (i.e., pixels at the lowest layer) through a set of channels (or *kernel*s) that represent the local spatial features.
- Since each convolution layer operates on the output of the previous convolution layer, higher layers correspond to increasingly complex features obtained by combining lower-complexity features.
- Since relevant features of interest can be of different sizes, *pooling/subsampling* layers are introduced among convolution layers: these pooling layers carry out down-sampling of the output of a convolution layer, thereby (given a fixed kernel size) effectively doubling the size of the feature extracted by the corresponding filter.

Intuitively, a CNN searches for increasingly complex local features that can be used for understanding (and interpreting) the content of a dataset. Such latent features (*deep representations*) are fundamental to the success of the deep neural networks, as each layer in the network sequentially feeds on the latent features (output) of the previous layers to learn rich and abstract features. More formally, a neural network ($\mathcal{N}$) is a sequential arrangement of layers ($\mathcal{L}$), mainly convolutional and dense layers, to map the input $\boldsymbol{X}$ to output $\boldsymbol{Y}$ as follows:

$$\boldsymbol{Y} = \mathcal{N}(\boldsymbol{X}) = \mathcal{L}_L(\mathcal{L}_{L-1}(\ldots\mathcal{L}_2(\mathcal{L}_1(\boldsymbol{X})))); \tag{5.1}$$

here, $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ and $\boldsymbol{Y} \in \mathbb{R}^{N\times O}$ where N is the number of samples, D is the dimensionality of the sample, O is the number of class labels, and L is the number of layers. Any given layer $\mathcal{L}_l$ can be generalized (*perceptron*) as,

$$\mathcal{L}_l(\boldsymbol{X}_l) = \sigma_l(\boldsymbol{W}_l\boldsymbol{X}_l + \boldsymbol{B}_l), \tag{5.2}$$

where $\boldsymbol{X}_l$ (the output of layer $l-1$, s.t. $\boldsymbol{X}_l = \boldsymbol{Y}_{l-1}$) is the input to the layer l (for $l = 1$, $\boldsymbol{X}_1 = \boldsymbol{X}$) and σ_l, $\boldsymbol{W}_l$, and $\boldsymbol{B}_l$ are the layer's activation function, weight, and bias respectively. Note that, if the l^{th} layer has m_l neurons and the $(l-1)^{th}$ layer has n_l neurons, then $\boldsymbol{Y}_l \in \mathbb{R}^{m_l\times 1}$, $\boldsymbol{X}_l \in \mathbb{R}^{n_l\times 1}$, $\boldsymbol{W}_l \in \mathbb{R}^{m_l\times n_l}$ and $\boldsymbol{B} \in \mathbb{R}^{m_l\times 1}$.

5.4.1.2 Attention Masks

As discussed in Sections 5.2.1 and 5.3, several researchers noticed that significant amount of waste in learning and inference effort can be avoided if the attention is directed towards parts of a data that are likely to contain interesting patterns. This is achieved by attaching so called *attention modules* to this neural architecture, where the output of the attention module is used to weight the features learned in the CNN [116, 83, 119]. Such attention mechanisms have shown to help improve the

network performance by facilitating the network with the ability to learn to *highlight* important and *suppress* unimportant features.

In CNNs, attention is achieved through the introduction of attention masks. As visualized in Figure 5.1, the layer contains an additional component called *attention mask* ($\boldsymbol{M}_l^a$):

$$\mathcal{L}_l^a = \mathcal{L}_l \odot \boldsymbol{M}_l^a \langle \boldsymbol{Y}_l \rangle. \tag{5.3}$$

Here, $\boldsymbol{M}_l^a$ highlights the important local regions in the image, and/or suppresses the unimportant regions. However, conventional attention mechanisms fail to consider information that a cross-scale examination of the latent features may reveal, and, this work argues that salient changes in the scale-space can be identified through a cross-scale examination of the latent features and the extrema in these changes can be leveraged for more effective attention masks.

5.4.2 Feature Search in Scale-Space

In the literature, there are several localized feature extraction algorithms for images: these include SURF [8], HOG [29], and SIFT [76]. In particular, SIFT has been the *de facto* representation strategy for content-based image retrieval as these features have shown robustness against rotation, scaling, and various distortions. Intuitively, each feature corresponds to a region in a given image that is *different* from its neighborhood, also in different image scales. These stable patterns are extracted through a multi-step approach, including (a)scale-space construction, (b) candidate key-point identification, (c) pruning of poorly localized, non-robust features, and (d) descriptor extraction.

The scale-space used for feature search is constructed through an iterative smoothing process, which uses Gaussian convolutions to create different versions of the input data, each with different amount of detail. Robust localized features

are then located where the differences between neighboring regions (possibly in different scales) are large – in other words, these keypoints are located at the *local extrema* of the scale space defined by the difference-of-Gaussian (DoG) of the input image. More specifically, an l-layer state space of an input image, $\mathbb{I}$, is defined as a set of data matrices $\{\boldsymbol{I}_0, \ldots, \boldsymbol{I}_L\}$, where $\boldsymbol{I}_l = \boldsymbol{I}\{\sigma_0 \times k^l\}$, is a smoothed version of the input image for some smoothing parameter σ_0 and a scaling parameter $k > 1$. Given this, a DoG, $\mathbb{G}$, is created by considering a sequence of difference matrices $\{\boldsymbol{D}_0, \ldots, \boldsymbol{D}_{L-1}\}$, where $\boldsymbol{D}_l = |\boldsymbol{I}_{l+1} - \boldsymbol{I}_l|$ and feature candidates are sought at the local maxima and minima of the resulting DoG: each $\boldsymbol{D}_l[x, y]$ (where x and y are the rows and columns, respectively) is compared against its 26 $(= 3^3 - 1)$ neighbors (spatial neighbors in the scale l and neighboring scales $l-1$ and $l+1$) and the triplet $\langle l, x, y\rangle$ is selected as a candidate only if it is close to being an extremum among these neighbors[2].

5.4.3 Scale-Space Attention Networks (SAN)

Despite their success in object recognition and image search, SIFT features described above have recently been overshadowed by CNNs in many image recognition tasks [70, 102, 101, 57, 105]. Yet, as discussed earlier, this advantage of CNNs are subject to several constraints: most importantly, due to the large number of parameters that need to be learned from data, CNNs require a lot of data objects for training. Features likes SIFT, on the other hand, are (a) relatively cheaper to obtain and (b) since they encode the key domain knowledge "a robust feature is one that is maximally different from its immediate neighborhood both in space and scale" algorithmically, they do not require training data. In this section, a novel attention

[2]The number of neighboring triplets may be less than 26 if the triplet is at the boundary of the image or scale space.

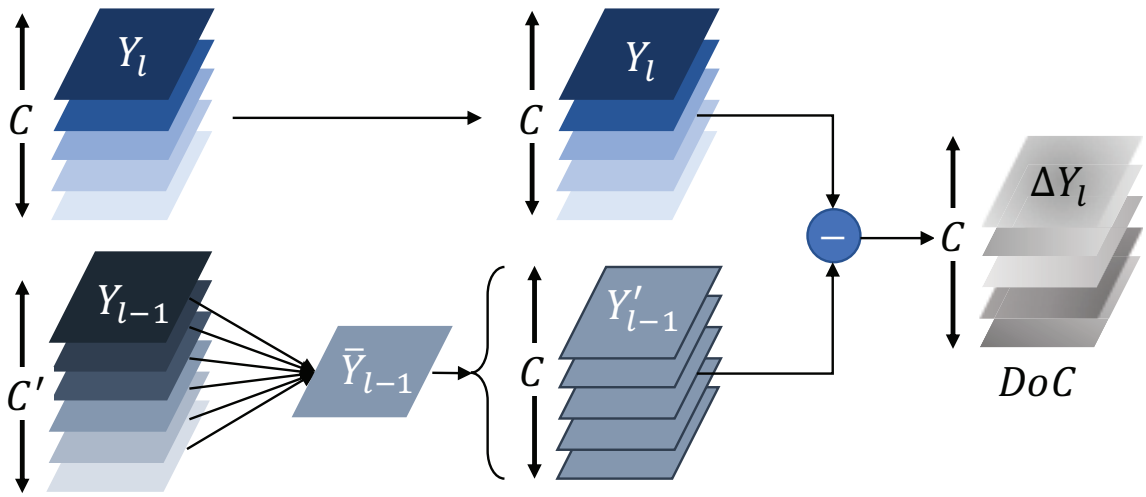


Figure 5.5: Overview of the Difference-of-Cconvolutions (DoC) Construction Module in SAN the Module Takes Latent Features ($\boldsymbol{Y}$) from Two Consecutive Layers ($l-1$ and l) and Transforms the Latent Features $\boldsymbol{Y}_{l-1}$ into the Same Basis Space as $\boldsymbol{Y}_l$ by Taking Average along the Channel Axis ($\overline{\boldsymbol{Y}}_{l-1}$), Followed by the Expansion of the Channel Dimension Through Replication to Obtain $\boldsymbol{Y}^{'}_{l-1}$; Finally, SAN Take the Absolute Difference ($\triangle \boldsymbol{Y}_l = |\boldsymbol{Y}_l - \boldsymbol{Y}^{'}_{l-1}|$) to obtain the DoC

module is constructed for CNNs based on a similar observation: "*the CNN should pay special attention to latent features that are maximally different from their immediate neighborhood both in space and scale*". In particular, unlike conventional attention mechanisms (Equation 5.3), SAN proposes to leverage outputs from two adjacent layers when constructing the attention module:

$$\mathcal{L}^a_l = \mathcal{L}_l \odot \boldsymbol{M}^a_l \langle \boldsymbol{Y}_l, \boldsymbol{Y}_{l-1} \rangle. \tag{5.4}$$

As discussed in the rest of this section, here $\boldsymbol{M}^a_l \in [0, 1]$, is a *soft*-attention mask obtained by identifying and augmenting the salient local regions within the latent features based on informative local changes. More specifically, SAN proposes to compare latent features (outputs) from two adjacent layers, $l-1$ and l to help identify

the robust salient region, as opposed to relying only an individual layer output, $\boldsymbol{L}_l$, as in conventional attention networks. Detail of the process are visualized in Figure 5.5, next:

5.4.3.1 Difference-of-Convolutions (DoC) Construction

Figures 5.4a and 5.4b show sample kernels learned in two consecutive layers. This chapter argues (and experimentally show in Section 5.5) that one can learn diverse attention masks by considering these two adjacent layers together. In order to extract salient regions in layer l, SAN first constructs a difference-of-convolutions (DoC) representation, which helps facilitate localization of scale-space changes that are prominent in an image. Note that unlike SIFT [76], where the DoG is constructed by performing a pixel-by-pixel subtraction of two Gaussian smoothed images, SAN, seeks the difference among the latent features $\boldsymbol{Y}_{l-1}$ and $\boldsymbol{Y}_l$, where the convolution kernels themselves are learned from the data. Therefore, the DoC is computed from the outputs of the two consecutive convolution layers, $l-1$ and l, as follows:

$$\triangle \boldsymbol{Y}_l = |\boldsymbol{Y}_l - \boldsymbol{Y}_{l-1}| \tag{5.5}$$

Experimental evaluation shows in Section 5.5, taking the absolute difference (as opposed to simple difference) has significant positive impact on the attention performance – this is because attention needs to be given to, not only maxima, but extrema of the difference of Gaussians. In addition, taking a non-absolute difference might cause multiple counter-intuitive effects in the network: first, the introduction of negative gradients may lead vanishing gradient problem as positive and negative gradients might cancel each other; secondly, the negative difference to *sigmoid* function will push the attention towards "0", as $sigmoid(x) \in [0, 0.5],\ \forall\ x \leq 0$.

Note, however, that there is a significant problem with the Equation 5.5: the latent features from the two layers do not have one-to-one correspondence, therefore the difference operation is not well defined: $\boldsymbol{Y}_l \in \mathbb{R}^{H \times W \times C}$ and $\boldsymbol{Y}_{l-1} \in \mathbb{R}^{H \times W \times C'}$, where H and W is height and width of the input image and C and C' are the number of channels/kernels in the layers, l and $l-1$, respectively. Therefore, the set of channels (kernels) $\mathbb{C}$ and $\mathbb{C}'$ where $C = |\mathbb{C}|$ and $C' = |\mathbb{C}'|$ potentially represent two different sets of basis vectors. Therefore, to implement DoC over these different sets of basis vectors, SAN proposes to take average along the channel dimension, s.t.

$$\begin{gathered} \overline{\boldsymbol{Y}}_{l-1}[h, w, 1] = \frac{1}{C} \sum_{c=1}^{C} \boldsymbol{Y}_{l-1}[h, w, c] \\ \forall h = 1 \dots H, w = 1 \dots W \end{gathered} \tag{5.6}$$

where $\overline{\boldsymbol{Y}}_{l-1} \in \mathbb{R}^{H \times W \times 1}$ represents the channel average of $\boldsymbol{Y}_{l-1}$. SAN, then, expands the channel dimension of $\overline{\boldsymbol{Y}}_{l-1}$ as

$$\boldsymbol{Y}_{l-1}^{'} = stack(\overline{\boldsymbol{Y}}_{l-1}, C'), \tag{5.7}$$

where C' is the number of channels of $\boldsymbol{Y}_l$ and the "$stack$" operation allows for stacking C' many replicas of $\overline{\boldsymbol{Y}}_l$ along the channel dimension to obtain $\boldsymbol{Y}_{l-1}^{'} \in \mathbb{R}^{H \times W \times C'}$. Consequently, the representative $\boldsymbol{Y}_l^{'}$ is now comparable to $\boldsymbol{Y}_l$ and the proposed attention mechanism, SAN, can be applied on two adjacent convolutional layers with different number of channels without a padding operation to align the dimensions.

Given the above, SAN defines the salient change across the latent features (updating the Equation 5.5) as follows:

$$\triangle \boldsymbol{Y}_l = |\boldsymbol{Y}_l - \boldsymbol{Y}_{l-1}^{'}|. \tag{5.8}$$

Here, $\triangle \boldsymbol{Y}_l$ represents the change in latent features defined in terms of the absolute difference between the two latent features. Samples results are presented in Fig-

ure 5.4c: as one can see in the figure, the DoCs discovered using two consecutive CNN layers retain large degrees of detail.

5.4.3.2 Extrema Detection

The SAN attention mechanism leverages the computed values of $\triangle \boldsymbol{Y}_l$ to learn the attention mask $\boldsymbol{M}_l^a$; but, one cannot use DoC directly as an attention mechanism: One reason for this is that the DoC itself can be subject to noise. This problem can be resolved by using the extrema of DoC rather than the DoC itself. However, simply detecting an extremum by exploring the neighborhood and marking it as "1" if it is a local extremum and "0" if not, might lead to a salt-and-pepper noise in attention, severely limiting the network's learning ability. Since the goal is to focus on the changes that are robust and prominent, SAN instead propose a weighted extrema detection mechanism, as follows:

$$\begin{aligned} \forall h = 1 \ldots H, w = 1 \ldots W, c = 1 \ldots C \\ \boldsymbol{Y}_l^e[h, w, c] = \alpha_{h,w,c} \times \triangle \boldsymbol{Y}_l[h, w, c], \end{aligned} \tag{5.9}$$

where

$$\begin{aligned} \forall h' \in \{h-1, h, h+1\},\ w' \in \{w-1, w, w+1\} \\ \alpha_{h,w,c} = \frac{\#of \triangle \boldsymbol{Y}_l[h, w, c] \geq \triangle \boldsymbol{Y}_l[h', w', c]}{9}. \end{aligned} \tag{5.10}$$

Here, $\alpha \in (0, 1]$ is the weighing parameter representing the portion of the DoC neighborhood (3×3 region around the coordinates $\langle h, w \rangle$) for channel c in layer l smaller than the DoC value $\triangle \boldsymbol{Y}_l[h, w, c]$. As seen in the degree of contrast present in the sample results in Figure 5.4d, this step helps highlight the salient points in the DoC while suppressing non-informative regions – the use of localized weighing *suppresses* noisy perturbations and retains more salient latent changes.

5.4.3.3 Extrema Smoothing

While the soft extrema detection mechanism on $\triangle \boldsymbol{Y}_l$ proposed above allows for *highlighting* salient changes and *suppressing* the noise through localized weighing, this operation can still leak certain amount of noise and artifacts in the weighed output, $\boldsymbol{Y}_l^e[h, w, c]$. Therefore, SAN further propose to leverage trainable convolutional layers, with kernels the same size as the kernels (k) of the feature extraction branch of the network, to smooth away such artifacts:

$$\boldsymbol{Y}_l^a = \sigma\left(\boldsymbol{Y}_l^e * \boldsymbol{W}_l^k + b_l^k\right) \tag{5.11}$$

The application of this additional convolutional layer acts as a blurring operation that smooths the unintended extrema artifacts, thus enabling the learning of a more robust attention mask. Sample results are presented in Figure 5.4e – note that, the smoothing operation, not only eliminates artifacts, but also boosts diversity relative to the pre-smoothed version of the extrema. The validity of this observation and the positive impact of this additional smoothing step are validated in the experimental evaluation section (Section 5.5, Table 5.9).

5.4.3.4 Attention Mask Recovery

In the final step of SAN, the convolved output is passed, $\boldsymbol{Y}_l^a$, through the *sigmoid* function to learn the final attention mask, $\boldsymbol{M}_l^a$, highlighting the salient attention regions:

$$\boldsymbol{M}_l^a = sigmoid(\boldsymbol{Y}_l^a). \tag{5.12}$$

Intuitively, the sigmoid function takes a real-valued vector of attentions and maps them to values in the range $[0, 1]$ such that entries in the vector that are away from 0 are saturated to 0 or 1 depending on whether they are negative or posi-

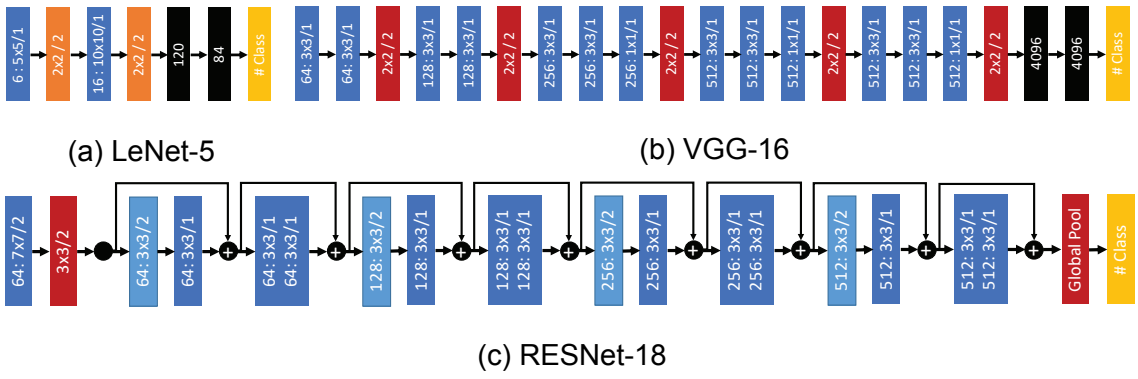


Figure 5.6: Overview of the Network Architectures for LeNet-5 [67], VGG-16 [106], and RESNet-18 [46]. Colors Represent, Blue: Convolution (stride=1), Light-Blue: Convolution (stride=2), Orange: Avg-Pooling (avg), Red: Max-Pooling (max), Black: Fully Connected (fc), and Yellow: Output Layer (fc-softmax)

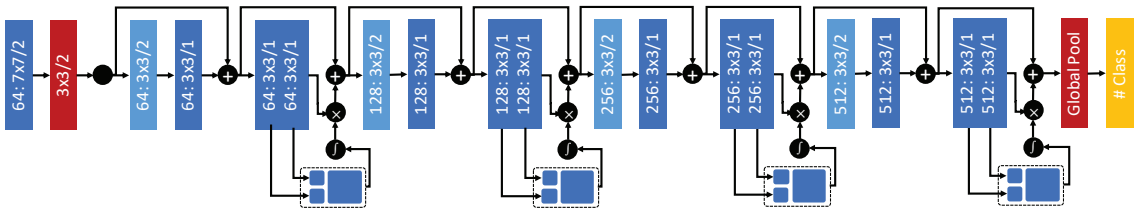


Figure 5.7: RESNet-18 with Bottleneck Attentions (Attention Applied Before Pooling Layers)

tive, respectively, and entries ~ 0 take a non-boundary value between 0 and 1 following a sigmoid shape. Consequently, $\boldsymbol{M}_l^a$ serves as a soft attention mask s.t. $\boldsymbol{M}_l^a[h, w, c] \in [0, 1]$ for layer l.

Figure 5.4f illustrates the rich (detailed and diverse) attention masks learned by the proposed scale-space attention network, SAN, as it intelligently uses the outputs of two adjacent convolutional layers to discover salient local regions to focus the attention.

5.5 Experiments

In this section, the proposed SAN framework is experimentally evaluated and is compared against the baseline, non-attentioned networks (LeNet-5 [67], VGG [106] and ResNet [46] - see Section 5.5.2) as well as the major competitors (CBAM [119], BAM [91], and RAN [83] (see Section 5.5.4 for more details) in bottleneck and full positions (5.5.3).

SAN was implmented in Python environment (3.5.2) using Keras Deep Learning Library (2.2.4-tf) [22] with TensorFlow Backend (1.14.0) [2]. All experiments were performed on an Intel Xeon E5-2670 2.3 GHz Quad-Core Processor with 32GB RAM equipped with Nvidia Tesla P100 GPU with 16 GiB GDDR5 RAM with CUDA-10.0 and cuDNN v7.6.4[3].

5.5.1 Datasets

- For the simpler LeNet network, data sets recorded in controlled environments were considered:
  - **MNIST** contains $60k$ and $10k$ training and testing handwritten digit images of 28×28 resolution [66].
  - **FMNIST** contains $60k$ and $10k$ training and testing images of 28×28 resolution with 10 classes [120].
- For the more complex VGG/ResNet Network, data sets recorded in real-world settings were considered:
  - **CIFAR10/20/100** contains $50k$ training and $10k$ testing images, respec-

[3]Results presented in this chapter were obtained using NSF testbed: "*Chameleon: A Large-Scale Re-configurable Experimental Environment for Cloud Research*"

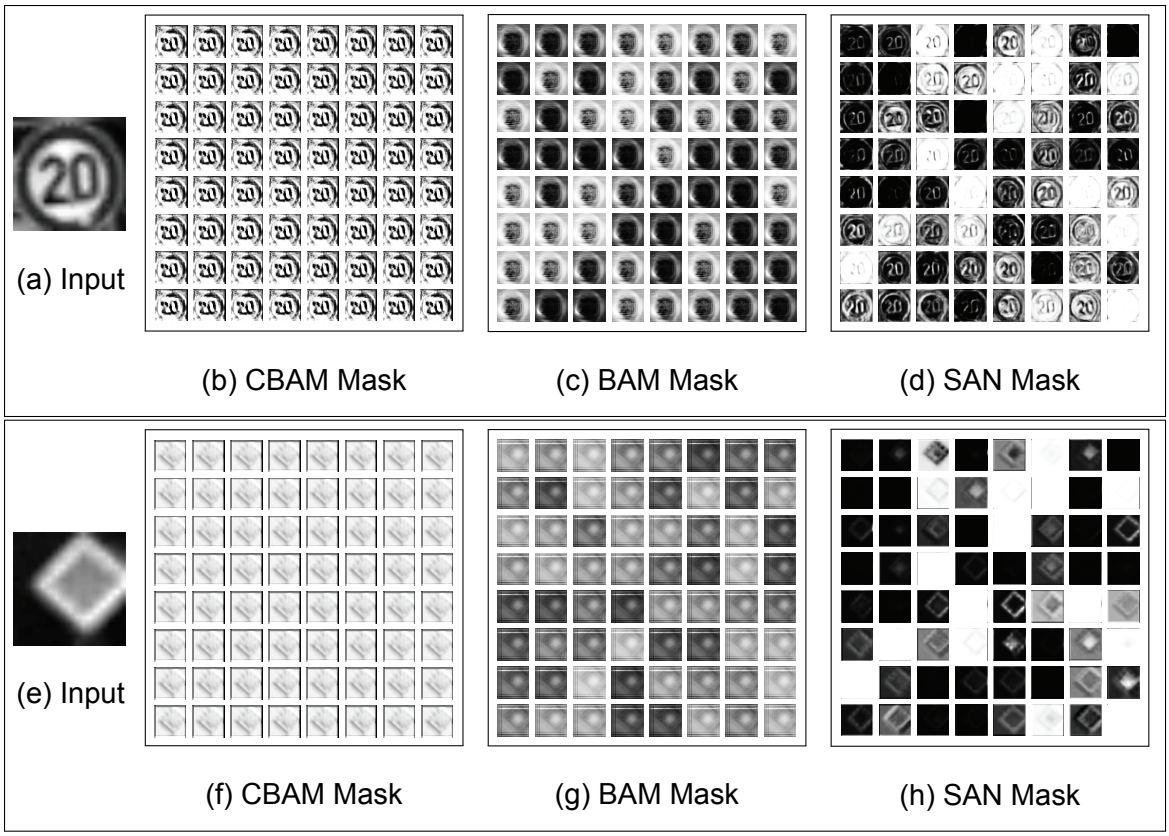


Figure 5.8: Attention masks learned by CBAM, BAM and SAN module for GTSRB dataset when placed at the *first* bottleneck position in VGG-16: SAN masks are more diverse and retain finer details from the input images

tively, with 32×32 resolution and the dataset contains 10, 20, and 100 labels [61].

- **GTSRB** dataset contains 39,209 and 12,630 training and testing images for 43 unique traffic sign [109].
- **GTSDB** is an object detection dataset with bounding boxes representing the positions of signs [49].
- **ImageNet** contains $\sim$1.23 million images for 1K real-world entities, with $\sim$1K images per entity [26].

- For recurrent networks, multi-variate time series data sets were considered:
  - **Mocap**: contains sensor (62) recording for 184 subjects for 8 gestures [38].
  - **FFC**: constains flight statistics for fuel usage, including temperature and wing position for 500 flights.

5.5.2 Baseline (Non-Attentioned) Architectures

5.5.2.1 LeNet-5

Designed for recognizing handwritten digits [67], LeNet-5 is a relatively simple network with 5 trainable (2 convolution and 3 dense) and 2 non-trainable layers using average pooling (Figure 5.6a). LeNet demonstrated that localized image features (handcrafts) can be substituted by deep features through the use of back-propagation of the classification error. The two convolution layers contain 6 and 16 kernels and dense layers have 120 and 84 kernels. Hidden layers are *tanh* and the final layer is *softmax*. LeNet's simplicity has made it the benchmark architecture for datasets recorded in constrained environments, such as MNIST, and FMNIST, in many works [55, 23, 35].

5.5.2.2 VGG

With the increase in complexity of data [26], a deeper and more complex architecture was required. An answer to this requirement was the VGG network [106] (Figure 5.6b), a 16 and 19 layer networks with 13 and 16 convolution layers respectively and 3 dense layers, with interleaved 5 max-pooling layers. VGG demonstrated that small kernel sizes (e.g. 3×3) can achieve better accuracies than using large kernels (e.g. 5×5 or 11×11). Furthermore, VGG leverages ReLU as the hidden activation to overcome the problem of vanishing gradient, as opposed to *tanh*. Furthermore,

the network uses a convolutional layer with kernel size 1×1 as 7^{th}, 10^{th} and 13^{th} layers in the network to introduce additional non-linearity and uses rectification operation. In addition to the kernel size, VGG proposed to slide the convolution kernel by 1 unit in each spatial direction and pooling kernel by 2 units along each spatial dimension. Given the ability of VGG network to learn the complex pattern in the real-world dataset, SAN used the network on datasets, such as CIFAR10/20/100, GTSRB, and ImageNet that contains complex, real-world objects.

5.5.2.3 ResNet

As seen in Section 5.2, much of the success of neural networks lies in their *depth* and *width*, however, as [44] shows, the network saturates, and may even degrade, after a certain depth is reached. ResNet [46] demonstrated that the problem of accuracy saturation/degradation might be alleviated by the use of *residual connections* (Figure 5.6c). SAN considered ResNet architecture for two depths: 18 and 50. For instance, ResNet-18 consists of 17 convolutional layers with varying (64, 128, 256, and 512) number of convolutional kernels, a 2D maxpooling layer, a global average layer, and a fully connected layer. First convolutional layer uses a kernel size of 7×7 and the remainder use 3×3 as kernel size. Each convolutional layer is followed by batch-normalization [55] and ReLU [86].

5.5.2.4 LSTM-4

SAN considered a 4 layers LSTM architecture comprised of 64 LSTM units each, with average pooling following every two recurrent layer, and final dense layer with *softmax* (classification) and *linear* (forecasting) as output activation. SAN was compared to recurrent attention network (RAN) [83].

5.5.3 Positioning the Attention Modules

As noted in Equation 5.1, and seen in Figure 5.6, a neural network is a sequence of layers interleaved by sub-sampling (pooling) layers. This means that there are multiple locations in the network where the latent features are being generated and transfered forward. As discussed in Section 5.4.1, SAN considered two alternative attention strategies:

– *Bottleneck placement strategy*: attention modules are applied before the data is down-size at pooling layers.

– *Full placement strategy*: in this case, attention modules are placed for every trainable layer in the network.

Figure 5.7 shows the version of the ReSNet-18 networks extended with attention modules, under bottleneck strategy.

5.5.4 Competitors

In this section, SAN was compared against the following competitors:

– *Non-Attentioned Baselines*: As the basic baseline, networks without any attention module were considered. In particular, *four* types of baseline architectures were explored: LeNet-5, VGG, RESNet, LSTM-4 (see Section 5.5.2 for more details).

– *Convolutional Block Attention Network (CBAM)*: CBAM is an attention module [119] that is designed to leverage contextual relationships among channel and spatial latent features to learn the attention mask. This is achieved by sequentially considering channel attention followed by spatial attention. Intuitively, the channel attention helps learn "*meaningfulness*" of the image,

followed by spatial attention to learn "*where*" this meaningful information lies in the image. [119] suggested that the CBAM modules are placed after convolutional layers.

– *Bottleneck Attention Module (BAM)*: In contrast to the CBAM's sequential approach towards learning channel and spatial attention, BAM [91] computes channel and spatial attention simultaneously, similar to inception networks [110]. BAM creates *three* branches in the network, 1) feature branch, 2) channel branch, and 3) spatial branch. In the *feature* branch, the latent features are propagated forward, similar to the conventional networks, whereas channel and spatial branch learn the respective attention masks. Note that unlike CBAM (which relied on conventional convolution layers), BAM used dilated convolution layers. BAM recommended that the attention modules be placed before the bottleneck.

– *Recurrent Attention Network (RAN)* [83]aims at learning the subset of input feature at t while relying on the model output at time *t-1*.

As described above, CBAM and BAM use different (full vs. bottleneck) attention module placement strategies. Both strategies were consderied when comparing SAN against the competitors. Figure 5.8 displays bottleneck attention for three sample images under CBAM, BAM, and the proposed SAN attention mechanisms.

Note also that CBAM and BAM rely on different (conventional and dilated) types of convolution layers. Therefore, SAN was trained with two different versions of the proposed attention modules: SAN-c with conventional convolution layers and SAN-d with dilated convolutional kernels.

Table 5.1: Model Classification Accuracies (Top-1 and Top-5) for ImageNet Data for VGG-16/RESNet-18 Model Architecture

	VGG-16				**RESNet-18**			
	Bottleneck		**Full**		**Bottleneck**		**Full**	
Datasets	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Base Model	71.90	90.6	71.90	90.60	70.40	89.45	70.40	89.45
CBAM	72.40	90.97	72.43	91.25	70.95	89.63	70.73	89.91
BAM	72.89	92.46	73.06	92.96	71.12	89.99	71.35	90.45
SAN-c	73.01	93.24	73.87	93.58	71.64	91.45	71.88	91.53
SAN-d	**73.57**	**93.97**	**74.26**	**94.07**	**72.01**	**92.87**	**72.38**	**92.93**

5.5.5 Experimental Results

5.5.5.1 Classification Accuracy

To evaluate the effectiveness of SAN framework, and demonstrate its robustness to the network architecture, in this section measures classification accuracies on *three* network architectures (LeNet-5, VGG-16, and RESNet-18) and on *seven* benchmark datasets (MNIST, FMNIST, CIFAR10/20/100, GTSRB and ImageNet)[4]. Top-1 ans Top-5 Classification accuracy results are presented in Tables 5.1, 5.4, 5.2, and 5.3. SAN defines the top-k accuracy as the ratio of the experiments in which the true class label is observed among top-k candidate class labels.

Figure 5.1 shows top-1 and top-5 classification results for the complex ImageNet dataset for VGG-16 and RESNet-18 network architectures. As observable in this figure, SAN-c and SAN-d consistently outperform the baselines and the attention

[4]Two types of SAN models were trained, first with conventional convolutions (CBAM) and SAN d with dilated convolutional kernels (BAM).

Table 5.2: Model Classification Accuracies (Top-1 and Top-5) for VGG-16 Model Architecture

	Bottleneck							
Datasets	**CIFAR10**		**CIFAR20**		**CIFAR100**		**GTSRB**	
Accuracy	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Base Model	72.70	93.12	45.17	70.01	31.07	53.01	96.21	99.84
CBAM	76.57	95.23	46.14	72.56	32.17	54.02	96.38	99.96
BAM	76.15	94.85	48.95	76.89	32.96	55.65	96.85	100.00
SAN-c	78.42	97.86	50.23	78.99	34.58	58.99	97.96	100.00
SAN-d	**79.01**	**99.50**	**52.84**	**82.03**	**36.14**	**61.23**	**98.25**	**100.00**
	Full							
Datasets	**CIFAR10**		**CIFAR20**		**CIFAR100**		**GTSRB**	
Accuracy	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Base Model	72.70	93.12	45.17	70.01	31.07	53.01	96.21	99.84
CBAM	74.65	94.26	45.74	72.16	32.51	54.67	97.45	100.00
BAM	76.42	95.63	46.79	73.68	35.96	61.45	97.73	100.00
SAN-c	79.89	97.99	51.14	82.99	37.59	63.48	98.31	100.00
SAN-d	**81.24**	**99.98**	**54.88**	**86.48**	**39.03**	**68.45**	**98.95**	**100.00**

competitors, CBAM and BAM. Figure shows that the results are relatively comparable for bottleneck and full strategies and also that the version of SAN which uses dilated convolutional kernels provides the overall highest accuracy gains under all scenarios. On the average, the accuracy gains provided by SAN-d is $4.91\times$ the accuracy gains provided by CBAM and $1.68\times$ the accuracy gains provided by BAM over the baseline. BAM's inception-style approach of having independent parallel

Table 5.3: Model Classification Accuracies (Top-1 and Top-5) for RESNet-18 Model Architecture

	Bottleneck							
Datasets	**CIFAR10**		**CIFAR20**		**CIFAR100**		**GTSRB**	
Accuracy	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Base Model	68.55	90.51	49.32	73.00	37.83	62.95	97.85	99.98
CBAM	73.76	96.40	53.39	80.34	40.95	65.65	98.11	99.99
BAM	73.42	93.42	53.53	81.13	40.72	65.39	98.42	100.00
SAN-c	74.82	97.86	55.23	85.82	41.74	67.10	99.24	100.00
SAN-d	**78.89**	**99.97**	**56.53**	**87.26**	**41.49**	**66.95**	**99.76**	**100.00**
	Full							
Datasets	**CIFAR10**		**CIFAR20**		**CIFAR100**		**GTSRB**	
Accuracy	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Base Model	68.55	90.51	49.32	73.00	37.83	62.95	97.85	99.98
CBAM	74.40	97.42	51.73	75.89	40.74	67.97	98.52	100.00
BAM	72.67	96.71	54.77	82.64	40.49	67.01	98.95	100.00
SAN-c	75.03	98.99	52.74	78.95	44.14	72.43	99.53	100.00
SAN-d	**79.67**	**100.00**	**54.94**	**83.98**	**47.06**	**74.42**	**99.85**	**100.00**

branch allows for better summarization of contextual information into attention mask than CBAM, however, the approach of taking the global average and maximum to summarize entire spatial information into single value limits the performance gains. SAN leverages the salient changes in latent features to identify points of attention to outperform both of these competitors.

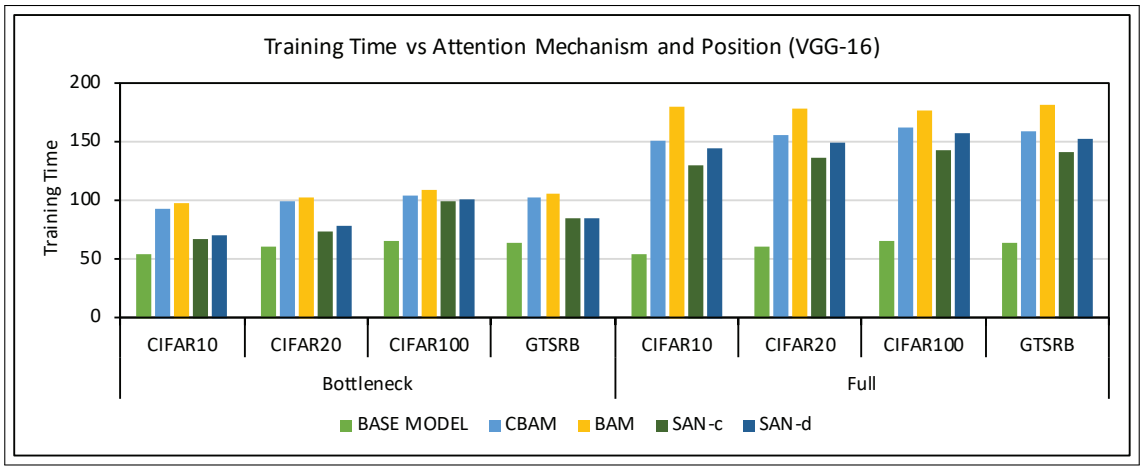
Table 5.4 presents the classification performance of different architectures on MNIST and FMNIST datasets with LeNet-5 architecture. Note that the LeNet-5 ar-

Table 5.4: Model Classification Accuracies (Top-1 and Top-5) for LeNet-5 Model Architecture

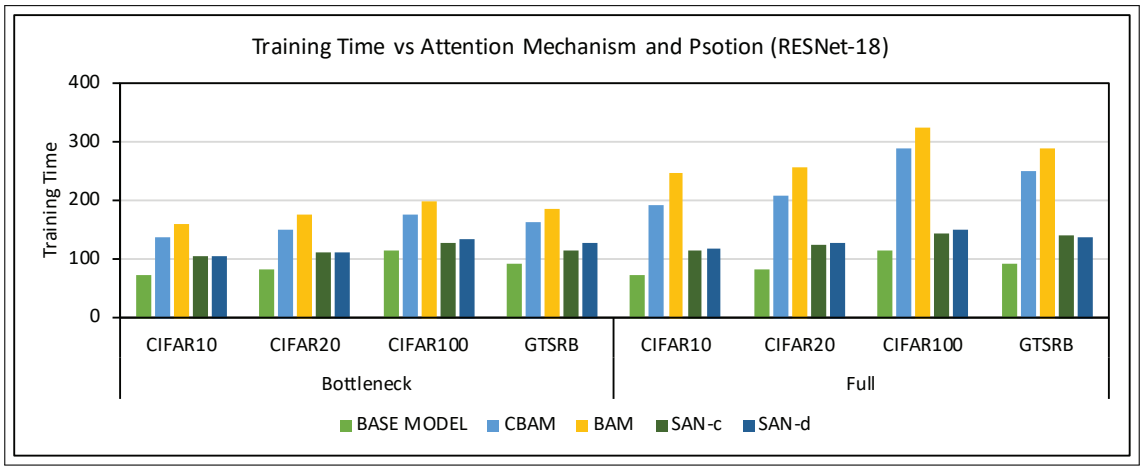
Datasets	**MNIST**		**FMNIST**	
Accuracy	**Top-1**	**Top-5**	**Top-1**	**Top-5**
Base Model	98.37	99.98	89.43	99.87
CBAM	97.88	99.98	89.27	99.85
BAM	98.52	99.99	89.64	99.90
SAN-c	98.56	99.99	89.75	99.92
SAN-d	**98.70**	**100.00**	**89.94**	**99.96**

chitecture is a special case, where the same architecture with SAN attention module represents both bottleneck and full attention model, as LeNet has only 2 convolution layers and each layer is followed by a down-sampling layer thus making it both bottleneck and full attention architecture simultaneously. Therefore, Table 5.4 does not present full and bottleneck results separately. As shown in the figure, thanks to the simplicity of the data, the baseline architecture has 98.37% (for MNIST) and 89.43% (for FMNIST) classification accuracy without any attention. Even in this scenario where there is very limited room for improvement, SAN-d improved the accuracy to 98.7% (for MNIST) and 89.94% (for FMNIST). In contrast, CBAM resulted in a drop in accuracy to 97.88% (for MNIST) and 89.27% (for FMNIST). Attention using BAM, on the other hand, provides some gains (98.52% for MNIST) and 89.64% for FMNIST), but lower than the gains provided by SAN-d.

In Tables 5.2 and 5.3, the application of different attention modules on VGG-16 and RESNet-18 architectures and their performance on relatively complex CIFAR10/20/100 and GTSRB datasets, under bottleneck and full attention placement



(a) VGG-16 Model Architecture



(b) RESNet-18 Model Architecture

Figure 5.9: Model Training Time (*in seconds*)

strategies is evaluared. These two figures reconfirm that, overall, SAN-d is the best attention strategy, providing significant up to 9.71% accuracy gains over the baseline.

Figure 5.8 provides sample attention masks to explain the key reasons behind the accuracy gains of SAN. As shown in this figure, SAN is able to learn rich (diverse and detailed) and robust attention masks. In stark contrast, CBAM learns only a single attention mask shared across all channels in the convolutional layer, severely

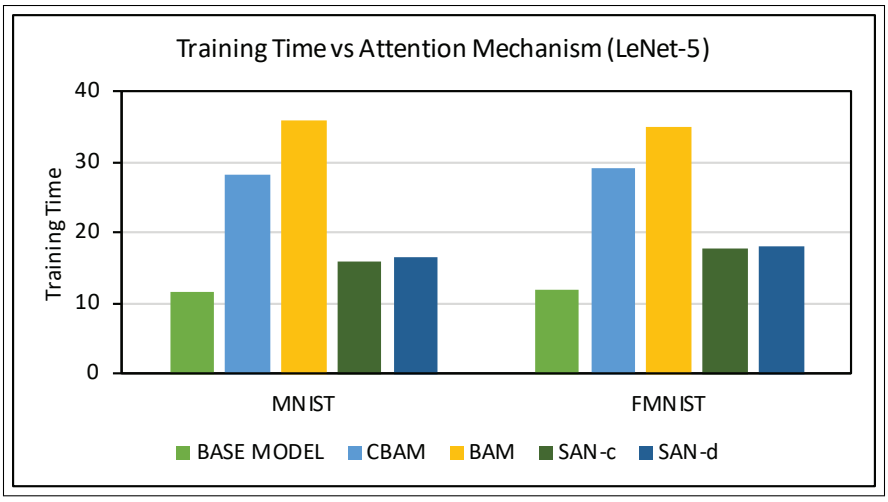


Figure 5.10: Model Training Time (*seconds*) for LeNet-5

limiting its ability to learn rich attention. While BAM does learn an explicit attention mask for each individual channel, it is able to retain coarser details - adding only limited richness to the network. In short, SAN's ability to account for the differences between two consecutive convolutional layers enables rich and robust attention masks leading to significant boost in network performance.

Table 5.5 demonstrates that SAN is able to outperform CBAM and BAM for deeper networks as well. SAN's ability to learn salient changes across layers proves beneficial, also when the level of abstraction increases in deeper networks.

5.5.5.2 Model Training Time

In Figures 5.9 and 5.10, the computational cost (training time) of SAN framework against the competitors are compared. The figures show that the proposed SAN mechanism introduces much smaller training overhead than the competitors, CBAM and BAM. While SAN-d, with dilated convolutional kernels, requires more training time than SAN-c, the difference is slight, and higher accuracy gains of SAN-d makes that difference worthwhile. One important observation comparing Fig-

ures 5.9a and 5.9b, is that on the same data, SAN provides significantly higher training execution gains over CBAM and BAM on (residual connection-based) RESNet-18 than on VGG-16. In fact, while the training cost for SAN strategies are similar for both networks, CBAM and BAM's training costs doubles when residual connections are introduced. This indicates that the scale-invariant robust attention generated through DoC extrema lead to a much more effective use of the residual connections. Overall, SAN provides models with higher classification accuracies compared to CBAM and BAM, at a significantly lower training cost.

5.5.5.3 Time Series Classification and Forecasting

In Table 5.6 and 5.7, one can see that SAN outperforms the base model as well as RAN for both classification and forecasting of multi-variate time series. For the classification task (Table 5.6), further observe that, while all three models are able to reach 100% model accuracy, SAN leads to 43% drop in model loss compares to Base model and 39% for RAN attention module for bottleneck positions, and for full attention position, SAN leads to 92% drop in loss against Base model and 89% against RAN, this demonstrated that the importance of input features learned (attention mask) by SAN is more informed and robust than the mask learned by RAN. For the forecasting task, SAN leads to maximum accuracy and minimum forecasting error[5].

[5] In Section 5.5.5.3 presents both model accuracy and loss. As in classification results, all models reach 100% accuracy, therefore, model loss is leveraged as a measure to compare the models and for forecasting, loss is used to measure the divergence of the forecasting results from ground truth.

Table 5.5: Classification Accuracy for Deeper Models - CIFAR10 (VGG-19 and ResNet-50) - Bottleneck

Network	**VGG-19**		**ResNet-50**	
Accuracy	Top-1	Top-5	Top-1	Top-5
Base	69.48	86.95	67.79	90.86
CBAM	70.24	89.85	65.82	89.74
BAM	71.41	94.01	68.76	94.51
SAN-c	77.14	96.21	71.36	96.89
SAN-d	**77.36**	**96.88**	**72.73**	**97.25**

Table 5.6: Model Classification Accuracy and Loss for MOCAP Dataset for LSTM Model[4]

Metric	**Accuracy**		**Loss** (MAE)	
Position	**Bottleneck**	**Full**	**Bottleneck**	**Full**
Base Model	100.00	100.00	0.1118	0.1118
RAN	100.00	100.00	0.1055	0.0751
SAN	100.00	100.00	**0.0641**	**0.0084**

5.5.5.4 Object Detection

Table 5.8 presents that for both bottleneck and full positions, SAN can better learn to detect objects of interest in an image. This highlights the importance of leveraging the salient changes across layers(while using only a single trainable layer in the attention module) as opposed to CBAM and BAM which rely on more than *one* trainable layers in each module.

Table 5.7: Model Forecasting Accuracy for Flight Fuel Consumption Dataset for LSTM Model[4]

Metric	**Accuracy** (cos. sim.)		**Loss** (MAE)	
Position	**Bottleneck**	**Full**	**Bottleneck**	**Full**
Base Model	0.9672	0.9672	40.88	40.88
RAN	0.9526	0.9745	37.04	33.40
SAN	**0.9745**	**0.9773**	**33.40**	**31.48**

Table 5.8: Model Object Detection Accuracy for GTSDB Dataset for VGG-16 Architecture

Position	**Bottleneck**	**Full**
Base Model	86.79	
CBAM	88.25	90.67
BAM	89.87	91.71
SAN-c	92.45	93.01
SAN-d	**93.88**	**95.63**

5.5.5.5 Ablation Studies

Finally Table 5.9 presents the ablation studies that validate the three key hypotheses underlying the SAN attention framework:

– *Cross-layer channel alignment*: As discussed in Section 5.4.3.1, the latent features from layers $l - 1^{th}$ and l^{th} layer are not represented on the same basis; therefore, SAN proposes an efficient transformation that maps these two layers onto a common basis, without padding. In Table 5.9, *w CCA* refers to case where channel alignment is used as described, whereas *w/o CCA*

Table 5.9: Model Classification Accuracy vs Model Architecture and Dataset Summarizing the Performance of Different Blocks Involved in Devising SAN Module ("–": Incompatible Configuration when Two Layers have Different Channel Counts)

Architectures		LeNet-5		VGG-16				RESNet-18			
Datasets		MNIST	FMNIST	CIFAR10	CIFAR20	CIFAR100	GTSRB	CIFAR10	CIFAR20	CIFAR100	GTSRB
0	**Base Model**	98.37	89.43	72.7	45.17	31.07	96.21	68.55	49.32	37.83	96.21
1	**no-Abs**, **no-CCA**, **no-Extrema**, **no-Smoothing**	–	–	68.83	39.48	22.07	93.25	65.82	46.36	41.53	93.51
2	**Abs**,**no-CCA**, **no-Extrema**, **no-Smoothing**	–	–	70.45	42.17	30.99	93.96	68.83	48.42	42.3	94.01
3	**Abs**,**CCA**, **no-Extrema**, **no-Smoothing**	98.34	89.55	73.95	40.19	32.34	95.98	71.53	49.17	42.86	96.23
4	**Abs**, **CCA**, **Extrema**, **no-Smoothing**	98.17	89.28	72.79	42.3	29.21	94.96	70.59	48.62	42.17	96.34
5	**Abs**, **CCA**, **Extrema**, **Smoothing**	**98.56**	**89.75**	**78.42**	**50.23**	**34.58**	**97.96**	**74.82**	**55.23**	**44.14**	**98.31**

refers to the case where channels are not aligned.

– *Absolute difference for extrema DoC construction*: As discussed in Section 5.4.3.1, SAN seeks attention at the extrema of the DoC – not only maxima – in order to prevent the "*sigmoid*" operation on the latent features to wipe-out heavily negative values, SAN defines DoC using absolute difference. In Table 5.9, *w Abs* refers to case where absolute differences are used to construct DoC, whereas *w/o Abs* refers to the case where simple (non-absolute) difference is used.

– *Attention to the extrema of DoC*: As mentioned above, to seek points of attention, SAN looks at the extrema of DoC. In Table 5.9, *w Extrema* refers to case where an extrema search step is applied on the DoC, whereas *w/o Extrema* refers to the case where the DoC is used directly without seeking its extrema.

– *Extrema smoothing*: As discussed in section 5.4.3.3, while extrema help iden-

tify the salient points in the latent features, smoothing of these extrema can help eliminate noise and improve robustness. In Table 5.9, *w Smoothing* refers to case where a final smoothing step is applied, whereas *w/o Smoothing* refers to the case where the smoothing step is omitted.

As shown in the table, the highest accuracies are obtained when all four steps are combined. It is especially interesting to see that, alone, extrema detection does not improve accuracy – results with extrema detection, but without smoothing (#4) are not better than results without extrema detection (#3); however, when combined with the final smoothing step (#5) to eliminate artifacts, extrema detection is very effective in boosting the overall accuracy.

5.6 Conclusion

This chapter presents the SAN framework a robust and model-independent attention module that aims to guide the attention of the network architecture to salient localized regions in the image/time series to boost the network accuracy, with minimal training overhead. To achieve this goal, an innovative robust feature learning framework is proposed with novel scale-invariant attention networks (SAN) that identify salient regions in the input data using extrema of the difference of Gaussians. Unlike the existing attention networks, SAN primarily concentrates attention on parts of the data where there is major change across space and scale. Experimental evaluation shows that the proposed attention module, SAN, can be successfully applied to various state-of-the-art architectures, such as LeNet-5, VGG-16, and RESNet-18, as an add-on to significantly boost the effectiveness, including for benchmark datasets. Experiments further showed that, SAN leads to minimal training overhead in comparison to the attention modules, such as CBAM, BAM, and RAN.

Chapter 6

SDMA: SALIENCY DRIVEN MUTUAL ATTENTION

6.1 Overview

Integration of rich sensory technologies into critical applications, such as gesture recognition and building energy optimization, has highlighted the importance of intelligent time series analytics. To accommodate this demand, uni-variate approaches have been extended for multi-variate scenarios, but naive extensions have lead to deterioration in model performances due to their limited ability to capture the information recorded in different variates and complex multi-variate time series patterns' evolution over time. Furthermore, real-world time series are often contaminated with noisy information. In this chapter, it is noted that a time series often carry robust localized temporal events that could help improve model performance by highlighting the relevant information; however, the lack of sufficient data to train for these events make it impossible for neural architectures to identify and make use of these temporal events. SDMA argues that a companion process helping identify salient events in the input time series and driving model's attention to the associated salient sub-sequences can help with learning a high-performing network. Relying on this observation, a novel Saliency-Driven Mutual Cross Attention (SDMA) framework[1] is proposed that extracts localized temporal events and generate a saliency series to compliment the input time series. Further, an architecture which accounts for the mutual cross-talk between the input and saliency

[1] Garg, Yash, and Candan, K. Selçuk, "SDMA: Saliency-Driven Cross Mutual Attention for Time Series Analytics", IEEE International Conference on Pattern Recognition, 2020

series branches where input and saliency series attend each other is proposed. Experiments show that the proposed mutually-cross attention framework can offer significant boosts in model performance when compared against non-attentioned, conventionally attentioned, and conventionally cross-attentioned models.

6.2 Introduction

Recent technological advances in sensory technologies have enabled large scale integration of sensor networks in a wide variety of applications, such as gesture recognition [34], weather monitoring [124], power usage forecasting [81], traffic sign detection [126] , and flood prediction [74]. This rapid integration, consequently, has lead to a significant explosion in the amount of temporal data being generated, both in terms of *depth* (length of time series) and *diversity* (type of time series). Sensor networks have enabled simultaneous recording a variety of attributes defining a system, leading to the generation of multi-variate time series - each variate corresponding to a different attribute being recorded. This explosion in the pace of multi-variate temporal data generation has stressed upon the importance of, intelligent time series analytics.

6.2.1 Time Series Analysis

With the increase in the availability of time series data, knowledge discovery tasks, such as classification and forecasting, that rely on these data have become increasingly more feasible. The problem of time series analysis often involves learning a function f than can map the observations recorded over time to a target output. This target output can be a discrete (classification), $f : \mathbb{X}_{1...T} \rightarrow \boldsymbol{Y}$, or a temporally evolving observation (forecasting), $f : \mathbb{X}_{1...T} \rightarrow \boldsymbol{Y}_{1...T}$. The ultimate goal of time series modeling is to learn from the past $(t_1, t_2, \ldots, t-1)$ observations and to

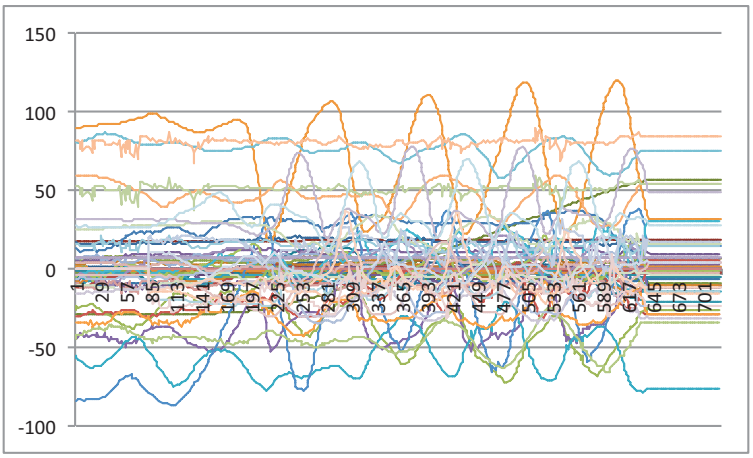


Figure 6.1: A Sample Multi-variate Time Series, Tracking 62 Sensors, Created by Body Motion Capture [1]

map the learning to the present (t) or the future ($t+1, \ldots, T$). The problem involves a set of recorded variables $\mathbb{X} \in \boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_d$ that drive a set target variables, $\mathbb{Y}$ (see Figure 6.1). For classification, $\mathbb{Y}$ the target variable is a set of class labels, and for forecasting, $\mathbb{Y}$ involves predicting the target variable sufficiently ahead in the future, which we refer as "*lead*" (l), i.e. $f : \mathbb{X}_{t-l} \rightarrow \mathbb{Y}_t$.

6.2.2 Neural Networks for Time Series Analysis

Many techniques, including Dynamic Topic Models (DTM) [12] and deep neural networks (DNNs), such as LSTMs [48], have been proposed to address time series imputation, labeling, and prediction tasks. LSTMs especially have been shown to be more effective than the conventional feed-forward and recurrent neural networks. Modern neural networks leverage *depth* and *width* of their models to learn complex patterns in the data in the form of deep features [110]. Conventional models, such as CNNs, and MLPs, lacks the ability to memorize temporal pattern over time. To counter this limitations, recurrent networks (RNN) introduced a memorize block in form of a recurrent connection to remember the pattern at time $t-1$ to help

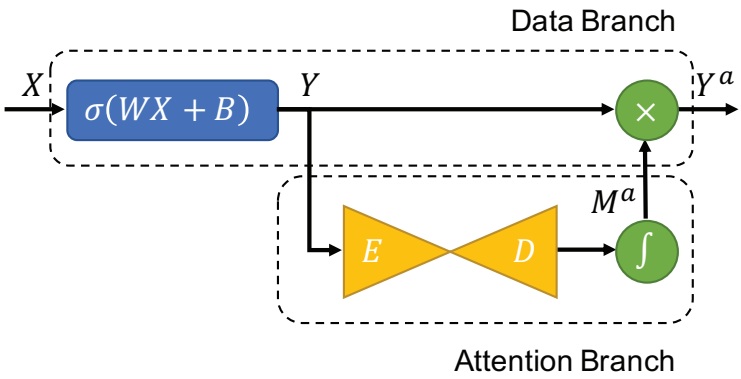


Figure 6.2: Overview of a Conventional Attention Mechanism, where an Attention Branch *Attends* on the Features Learned by a Data Branch

inform the network at time t [99]. Consequently, one difficulty with neural network based inference is the large number of model parameters that need to be learned from data. This is especially problematic for sparse and noisy data sets where it is difficult to learn these model parameters for accurate inference. Yet, as the number of variates and lengths of the time series increase, it is becoming increasingly difficult to learn an effective model, specially for recurrent models [7]. Existing solutions tend to fail when the events that are being inferred are rare or when detecting and predicting anomalies.

6.2.3 Attention Mechanisms

The quality of the time series models often suffer from the *curse of dimensionality*. In particular, the model accuracy depends on how well the model captures the richness and complexity of the patterns encoding in the variates. Recent research has shown that attention mechanisms (that help the neural network to focus on different aspects of the data at different stages of inference) has the potential to alleviate this difficulty to some degree. As [7] points out, a complementary atten-

tion block can help the network self determine the importance of the input features and highlight a subset of significant input features. In general, an attention network (see Figure 6.2), contains a feature and an attention branch interleaved together. Here the aim of attention branch is to feed on the feature branch output and learn the importance of individual deep features. However, it can be observed that a naive attention block might not lead to substantial gains in network performance (as shown in Section 6.5). DNNs capture features at different scales (*depths*), however these features remain vulnerable to noisy and redundant information, consequently leading to overfitting. Recently, a variety of multi-stream deep architectures that leverage cross attention to counter noisy and redundant information have been proposed [84, 3]. The challenge with such attention mechanisms, is that the attention model itself needs to be constructed carefully to ensure that the model focuses on the most relevant parameters, without mistakenly ignoring parameters critical for the inference task.

6.2.4 Key Contributions: SDMA Framework

In this chapter, it is noted that a time series often carry robust localized temporal events that could, at least in theory, help improve model performance; however, the lack of sufficient data to train for these features make it impossible for neural architectures and their attention mechanisms to identify and make use of these features.

Observation #1 (Localized salient event-driven input attention): Traditional attention mechanism assume that the networks can learn sufficiently rich deep features for the task at hand. However, this might not be the case. It can be argued that a separate process helping identify salient temporal events (highlighting the relevant temporal information) and driving model's attention to the associated salient

sub-sequences can help with learning a high-performing network. Recent works, such as [32] and [33], have shown that scale-space based approaches can be used to learn high-performing networks. In particular, even though localized features, like SIFT (Scale Invariant Feature Transform) [76] (for images) and UVTF (Uni-Variate Temporal Events) [15] (for univariate time series), may not lead to very accurate classifiers themselves, the information captured by these can nevertheless be useful to help support high performing networks. In particular, one can identify salient temporal events and their temporal scopes (sub-sequences) on a time series and focus the model training (attention) on the identified sub-sequences [15].

These salient points highlights the relevant information encoded in the input time series, which can be used to help improve the model performance.

Observation #2 (Mutually-supporting cross attention): Cross attention mechanisms, such as [77, 3, 84], help learn attention mask from one modality of time series to attend another – Therefore, in theory the saliency series can be used to learn attention masks for the input data series. However, SDMA argues argue (and experimentally observe) that this will provide limited gains, because any noise in saliency can potentially impact the overall accuracy. Therefore, SDMA argues that a mutually-supporting cross attention mechanism between the input and saliency branches ("attending the saliency" and "attending the input") can help achieve superior model performance, by capturing the relevant information (*in itself*) and simultaneously suppressing the noisy information (*in the other*) and vice versa.

Summary: In summary, **(a)** one can learn accurate multi-variate models by leveraging salient *multi-variate temporal events* and their temporal scope to direct models' attention to the salient information in the time series. **(b)** Instead of an architecture where the attention branch attending on the feature branch, an alternative architecture where both branches are attending on each other can lead to superior

performance. Relying on these observations and arguments, this chapter presents a novel *SDMA* Framework, which brings together the following core observations. Key contributions of SDMA are experimentally validated in Section 6.5 and observe that SDMA is able to learn a high-performing model by leveraging the localized temporal events and by implementing a cross-talk between feature and attention branch, and compare it against state-of-the-art attention networks, such as SAN [33] and DSTP [73] against their performance on various benchmarks datasets.

6.2.5 Organization of the Chapter

This chapter is organized as follows: Section 6.3 describes the existing work in detail, Section 6.4 presents the proposed RACKNet framework, and Section 6.5 discusses the performance and robustness of RACKNet framework under various settings involving change in kernel budget and dropout rate for both CNN and RCNN designs. Chapter conclude in Section 6.6.

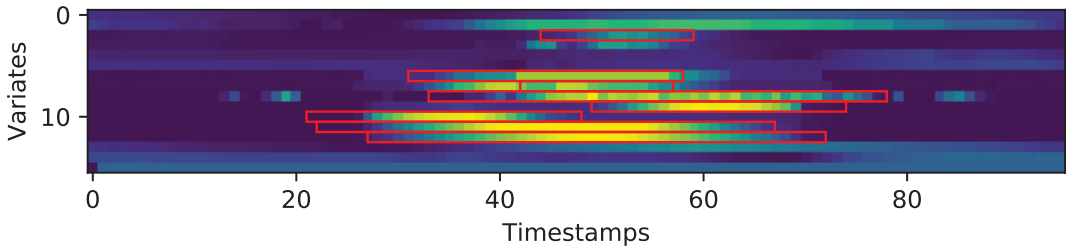
6.3 Related Works

Success of neural networks various domains, such as face recognition [64], power forecasting [81], traffic sign detection [126] , and flood prediction [74] have driven the shift-away from localized-event based approaches, which provides a rough overview through systematic extraction salient event points. Localized events extraction leverages multi-scale representation of the input data to extract events at different scale to capture scale-invariant information in the data by highlighting relevant and suppressing irrelevant information. [15] has shown that key-events can help reduce the computational cost of determining DTW warping path. [31] demonstrated that localized features can be used to reduce the computational cost of DTW-based similarity measures. More recently, [32, 33] used be used learn high-

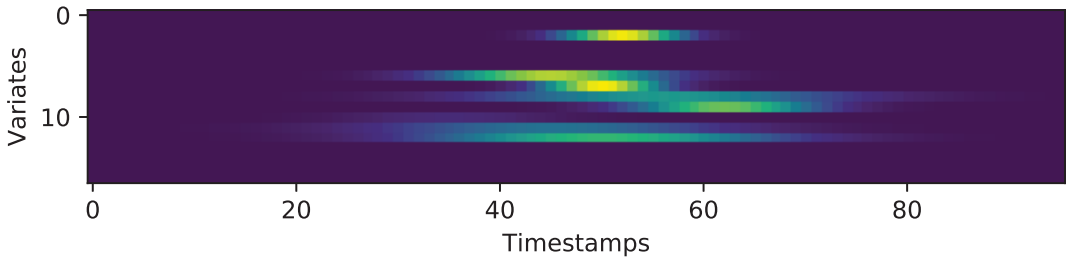
performing networks buy either using these event in pre-traiing analysis of data or emulating event extraction during model training. Recurrent networks (RNN) introduced a recurrent connection to remember the patterns at time $t-1$ to learn patterns at time t [99]. However, RNNs' suffers from catastrophic forgetting for long time series [7]. Long short-term memory (LSTM) extended RNNs' by using a cell state along with hidden state to robust memorization [48]. While LSTM has proven successful [73, 94, 114] they remain vulnerable to noisy learning for multivariate time series. [7] proposed attention mechanism as a measure to highlight relevant and suppress irrelevant features. DSTP [73] and DA-RNN[94] extended the attention by exploring inter-variate and individual temporal relationship in time series.

6.3.1 Cross-Attention Mechanism

With the rise in multi-modal learning, cross attention mechanisms have been readily observed in the works such as image-text matching. [54] showed that deep features when learned for each modality together in the single feed-forward phase, leads to a better model performance. However, this work did not allow the modalities to attend each other. [77] extends [54] by proposing a stack-cross attention module, where first the image attended the text(caption) following by attentioned text attending the image. [3] proposed to measure the degree of a disaster by using image and text perform a mutual cross attention to eliminate the noisy information, however, the input features to the cross attention modules were derived from pretrained DenseNet (image) and BERT (text) model. [84] proposed a cross attention mechanism where two image representation (HSI and LiDAR) of the same object were used to learn rich spectral and spatial attention masks to attend the deep features learned by the network. Each of these works, [54, 77, 3, 84] have shown



(a) Localized Temporal Events and Their Scopes (Red Boxes)



(b) Corresponding Saliency Series

Figure 6.3: (a) Localized Temporal Events Extracted on a Multi-Variate Time Series and (b) the Corresponding Saliency Series Learned using the SDMA Framework

that for their respective evaluation models with cross attention demonstrate a superior performance to the model that only had self-attention, thus highlighting the fact that different modalities capture different aspects of the data, and when each modality attended the other, they were able to highlight the relevant and suppress the irrelevant information.

6.4 SDMA Framework

As discussed in Section 6.2, advances in sensory technologies has enabled large-scale assimilation of streaming data (recording multiple dependent and independent attributes simultaneously - multi-variate time series) at a very large scale and speed. A multi-variate time series potentially contains both relevant and irrele-

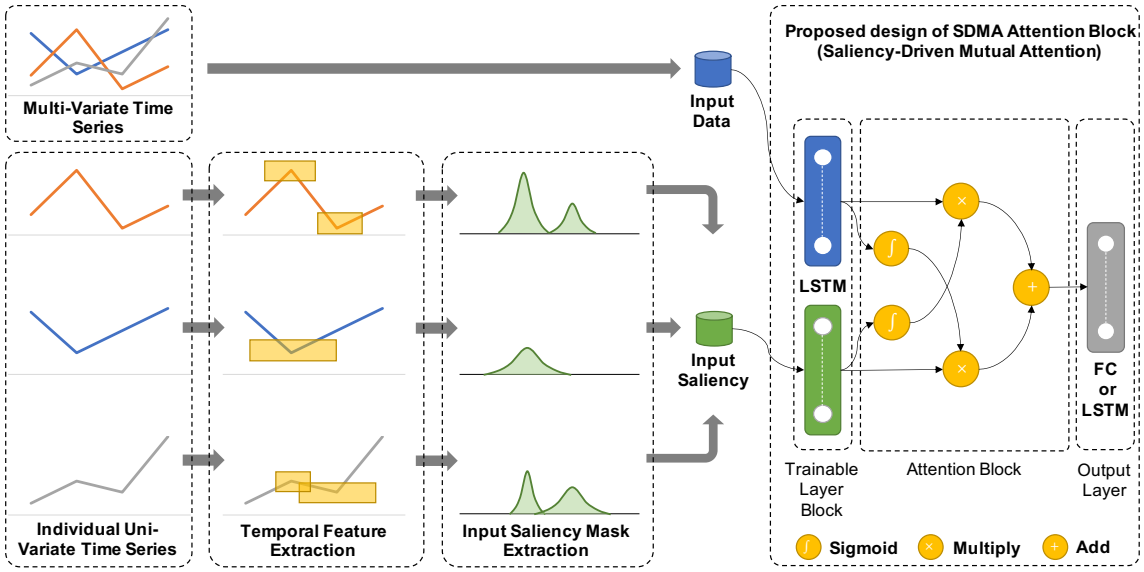


Figure 6.4: An Overview of the Proposed SDMA framework to Learn Saliency-informed Input Attention Mask and the Architectural Design of the Mutually-supporting Cross Attention Block: SDMA comprises of *Three* Stages: (1) Temporal Event Extraction, (2) Constructing Saliency Series, and (3) Model Training With Mutually Supporting Cross Attention. A Fully Connected (Fc) and Lstm Layer Is Used As the Output Layer for Classification and Regression Task Respectively. * Note That the SDMA block Can Be Extended to Multi-layer Architecture (As Shown in Figure 6.10)

vant information and, unless the focus (or attention) of the network is pulled towards relevant portions of the data, this can adversely impact the network performance. Therefore,the structural information of the variate (in the form of *salient temporal events*) can be leveraged to learn in an input attention to help highlight relevant and suppress irrelevant information in a given variate (Figure 6.3). In this section, the proposed *SDMA Framework* is presented that leverages salient events and their temporal scopes as a way to generate salient input attention for the neural network (Figure 6.4).

6.4.1 Uni- and Multi-Variate Time Series

A uni-variate time series (UVTS) is a sequence of ordered pair of observations and time at which observations were recorded for a given attribute (variate),

$$\boldsymbol{T} = \left[\left(\mathsf{v}_1, \mathsf{t}_1 \right), \left(\mathsf{v}_2, \mathsf{t}_2 \right), \ldots, \left(\mathsf{v}_T, \mathsf{t}_T \right) \right]. \tag{6.1}$$

While in general the temporal separation between two consecutive timestamps can be non-periodic, in this chapter the timestamps recorded in a UVTS are assumed to be periodic in nature.

A multi-variate time series (MVTS), $\mathbb{T}$, is a set of uni-variate time series, $\boldsymbol{T}$, s.t.

$$\mathbb{T} = \{ \boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_V \} \tag{6.2}$$

where, V is the number of variates, $\mathbb{T} \in \mathbb{R}^{V \times T}$, and $\boldsymbol{T} \in \mathbb{R}^{1 \times T}$, and T is number of timestamps.

6.4.2 Localized Temporal Events

Localized patterns have been shown to be effective for image retrieval [76] and motion classification [75] tasks. Similarly to these works, the localized event extraction process consists of (a) "scale-space generation" and (b) "extrema detection" steps to identify key intervals (or "events") in the individual variates. The first step of the process is to create a scale-space consisting of multiple smoothed versions of a given series – each resulting series are then subtracted from the series in the adjacent temporal scale to obtain difference-of-Gaussian series. Intuitively, the smoothing process can be seen as generating a multi-scale representation of the given series and thus the differences between smoothed versions of a given series correspond to differences between the same series at different scales. Based on

Figure 6.5: Creating the Gaussian Scale-Space and DoG for a Variate from a Multi-Variate Time Series. Here, y and x Represents the Variate and Temporal Dimensions of a Time Series

the argument that the interesting events will be maximally different from the overall pattern in their local neighborhoods, SDMA searches for those points that have largest variations with respect to both time and scale.

6.4.2.1 Temporal Scale-Space Generation

Figure 6.5 visualizes the process for creating the scale-space from a multi-variate time series. Let $\boldsymbol{T}_v$ represent a uni-variate time series, s.t. $\boldsymbol{T}_v \in \mathbb{T}[v, *]$, and $\boldsymbol{T}_v^{(t,\sigma)}$ represents the smoothed version of $\boldsymbol{T}_v$ through convolution with the Gaussian function along the temporal dimension:

$$\mathbf{G}\left(\mathrm{t}, \sigma\right) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-t^2}{2\sigma^2}} \tag{6.3}$$

such that

$$\boldsymbol{T}_v^{(\mathrm{t},\sigma)} = \mathbf{G}\left(\mathrm{t}, \sigma\right) \circledast \boldsymbol{T}_v, \tag{6.4}$$

where, $\circledast$ represents the convolution operaiton. Intuitively, Gaussian smoothing can be perceived as a multi-scale representation of a given series ($\boldsymbol{T}_v$), and the subsequent differences of the different Gaussian smoothed (Difference-of-Gaussian - DoG) series correspond to difference of the same series at difference scales. Therefore, the DoG can be computed as

$$\mathbf{D}_v^{(\mathrm{t},\sigma)} = \boldsymbol{T}_v^{(\mathrm{t},k\sigma)} - \boldsymbol{T}_v^{(\mathrm{t},\sigma)}. \tag{6.5}$$

Here, $\mathbf{D}_v^{(\mathrm{t},\sigma)}$ is the difference between the representation of the same input series ($\boldsymbol{T}_v$) smoothed at different scales, σ and $k\sigma$ (here, k is the constant multiplicative

Figure 6.6: A Sample Candidate Event Point, $\mathcal{F}$, (Solid Black) and its Neighbors in Adjacent Scales "$s+1$" (Red) and "$s-1$" (Yellow) and in Time "$t-1$" (Blue) and "$t+1$" (Green)

factor).

Analogous to SIFT, series is initially smoothed by a factor of σ_0, incrementally smooth the series by a multiplicative factor k. The scale space is organized as octaves, $\mathcal{O}$, where each of these $\mathcal{O}_o,\ o \in \{1, ..., O\}$ octaves is further organized as scales, $\mathcal{S}$ s.t. a given scale $\mathcal{S}_s,\ s \in 1, ..., S$ has an associated smoothing factor of $k^s\sigma$ s.t. $k^S = 2$, within each octave o:

$$\boldsymbol{T}_v^{t,2^o\sigma_0} = \boldsymbol{T}_v^{t,(k^S)^o\sigma_0} \tag{6.6}$$

As shown in Figure 6.5, for each octave (where the degree of smoothing is doubled), the length of the series is reduced (by half) for efficient computation, but this is not critical for the discussion.

6.4.2.2 Extrema Detection

In the next step, the points of interest are searched, $\langle v, t, o, s\rangle$ across multiple scales of the given time series, v, by – here o denotes an octave and s denotes the corresponding scale.

The search of local extrema (events) is performed by comparing the immediate neighbor (see Figure 6.6) along both time and scale in the DoG representation of

Figure 6.7: Overview of Event Scope and the Significance of Information Around the Event Center (•): the Significance of the Event Reduces as it Moves Further Away from the Center

the input series, $\boldsymbol{T}_v$. This helps identify events that are different from their local neighborhoods, both in scale and time.

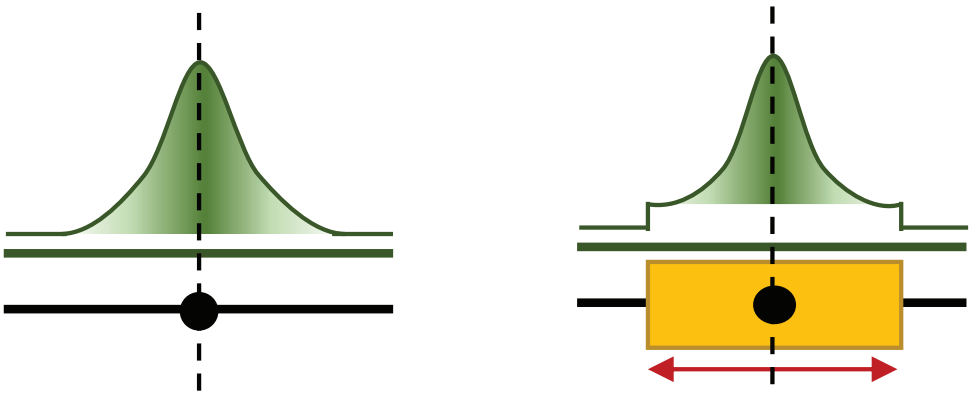
Note that, each identified event has a temporal scale, $s = (o \times S) + s$, defined by the octave o and scale s, in which it is located. More formally, a quadruple $\langle v, t, o, s \rangle$, is an extremum if it is maximum across its *eight* neighbors, *three* is each adjacent scales ($s-1$ and $s+1$), and *two* in time ($t-1$ and $t+1$)) i.e.

$$max \begin{pmatrix} \boldsymbol{D}_v^{t-1,k^{s+1}\sigma_0} & \boldsymbol{D}_v^{t,k^{s+1}\sigma_0} & \boldsymbol{D}_v^{t+1,k^{s+1}\sigma_0} \\ \boldsymbol{D}_v^{t-1,k^{s}\sigma_0} & \boldsymbol{D}_v^{t,k^{s}\sigma_0} & \boldsymbol{D}_v^{t+1,k^{s}\sigma_0} \\ \boldsymbol{D}_v^{t-1,k^{s-1}\sigma_0} & \boldsymbol{D}_v^{t,k^{s-1}\sigma_0} & \boldsymbol{D}_v^{t+1,k^{s-1}\sigma_0} \end{pmatrix} \tag{6.7}$$

In other words, $\langle v, t, o, s \rangle$ is designated as an extremum if it is greater than Θ% of the maximum of it's *8* scale-time neighbors in DoG ($\boldsymbol{D}$). Once the extrema have been identified, a threshold center amplitude criteria (intensity threshold) is imposed to ensure the the identified extrema has a minimum contrast of 0.005 as used in [76].

6.4.2.3 Event Representation

Finally, an event, $\mathcal{E}$, is defined as a $\langle v, t, \sigma, a \rangle$, where v is the variate on which it is identified, t is the center of the event, $\sigma = k^{(o \times S)+s} \times \sigma_0$ is its temporal scale, and a is the average of the observed values within 3σ (under Gaussian smoothing *three* standard deviation would cover ~ 99.73% of the original temporal points that has contributed to the event (Figure 6.7).



(a) Without-Cut (b) With-Cut

Figure 6.8: Two Strategies to Generate Attention Mask

6.4.3 Construction of the Saliency Series

In the next step, the localized temporal events identified on $\boldsymbol{T}_v$ and their event scopes are translated into a saliency series $\boldsymbol{S}_v$.

6.4.3.1 Per-Event Saliency

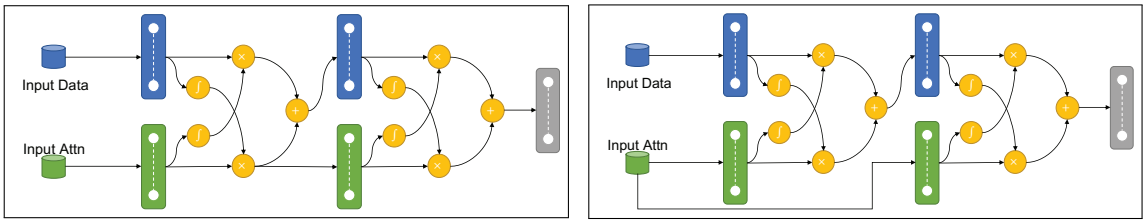
As seen in Section 6.4.2, a temporal event, $\mathcal{E}_i$, can be described using *four* essential attributes, $\langle v_i, t_i, \sigma_i, a_i \rangle$, where σ_i represents the standard deviation of the corresponding Gaussian kernel. Note that, by construction, the most significant information of a given event lies closer to the event's center, t_i, and this reduces on moving away from the event center (Figure 6.7):

$$\boldsymbol{M}_{v,i}(t, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{\frac{-(t-t_i)^2}{2\sigma_i^2}}. \tag{6.8}$$

Gaussian series is normalized to obtain the normalized series, $\boldsymbol{M}_{v,i}$:

$$\boldsymbol{N}_{v,i} = \frac{\boldsymbol{M}_{v,i}(t, \sigma_i)}{max(\boldsymbol{M}_{v,i}(t, \sigma))}. \tag{6.9}$$

Figure 6.9: Aggregate Attention Mask with Three Local Events



(a) SDMA Type 1 Architecture

(b) SDMA Type 2 Architecture

Figure 6.10: Overview of the *Two* Types of SDMA Architecture to Integrate the Attention Block in a Multi-Layer Network

Next the event amplitude, a_i, is incorporated to obtain the corresponding saliency series:

$$\boldsymbol{S}_{v,i} = a_i * \boldsymbol{N}_{v,i} \tag{6.10}$$

Note that, since it relies on a Gaussian function centered on the time t_i, the input mask defined above extends from the very beginning of the time series to its very end (Figure 6.8a). Alternatively, since under Gaussian smoothing *three* standard deviation would cover $\sim 99.73\%$ of the original temporal points that has contributed to the event, the start and end of the mask is defined as $t_i - 3\sigma_i$ and $t_i + 3\sigma_i$, respectively and the rest is set to 0 (Figure 6.8b). The two techniques are referred as saliency series *without cut* and *with cut*, respectively.

6.4.3.2 Aggregate Saliency

In the final step, the saliency series from all identified events are combined. In particular, let v be a variate and let $\mathbb{E}_v$ be the set of all events identified on this variate. The aggregated saliency series $\boldsymbol{A}_v \in \mathcal{R}^{1\times T}$ for this variate is computed as

follows (Figure 6.9):

$$\boldsymbol{A}_v = \sum_{\mathcal{E}_i \in \mathbb{E}_v} \boldsymbol{S}_{v,i}. \tag{6.11}$$

The saliency series of all variates are then passed to the next phase for model training with mutually supporting cross attention – in particular, these salience series are used to guide SDMAto attend on relevant parts of the series.

6.4.4 Mutually-Supporting Cross Attention

In this section, the design of SDMA attention block is proposed, which introduces mutually-supporting attention across feature and attention branches. As discussed in the introduction, localized events are rough approximations of salient information in the series. The richness of the attention masks inherently depend on the discriminatory power of the input events to the attention block – therefore, it can be argued that by enabling a bi-directional cross-talk between the feature branch and attention branch, in the form of mutually-supporting attention, can help overcome the limitations of both.

6.4.4.1 Single-Layer SDMA

As shown in Figure 6.4, the SDMA comprises of two LSTM layers, one for the input data and the other for saliency-based attention. The purpose of these two LSTMs is to learn deep representation of input series and saliency series, respectively, over time, through a combination of linear and non-linear operations. Unlike a conventional attention mechanism (Figure 6.2), however, while the attention branch attends the feature branch, the feature branch also attends the attention branch. Let $\boldsymbol{T}$ be a time series corresponding to variate v and $\boldsymbol{S}$ be the corresponding saliency series created through robust event discovery process describe above;

then the SDMA model can be described as

$$\boldsymbol{Y}_{\boldsymbol{T}_v} = LSTM(\boldsymbol{T}_v, \boldsymbol{W}, h, c) \tag{6.12a}$$

$$\boldsymbol{Y}_{\boldsymbol{A}_v} = LSTM(\boldsymbol{S}_v, \boldsymbol{W}, h, c) \tag{6.12b}$$

$$\boldsymbol{U}_{\boldsymbol{T}_v} = sigmoid(\boldsymbol{Y}_{\boldsymbol{T}_v}) \tag{6.12c}$$

$$\boldsymbol{U}_{\boldsymbol{A}_v} = sigmoid(\boldsymbol{Y}_{\boldsymbol{A}_v}) \tag{6.12d}$$

$$\boldsymbol{G}_{\boldsymbol{T}_v} = \boldsymbol{Y}_{\boldsymbol{T}_v} \odot \boldsymbol{U}_{\boldsymbol{A}_v} \tag{6.12e}$$

$$\boldsymbol{G}_{\boldsymbol{A}_v} = \boldsymbol{Y}_{\boldsymbol{A}_v} \odot \boldsymbol{U}_{\boldsymbol{T}_v} \tag{6.12f}$$

$$\boldsymbol{Y}_{o,v} = \boldsymbol{G}_{\boldsymbol{T}_v} + \boldsymbol{G}_{\boldsymbol{A}_v} \tag{6.12g}$$

Intuitively, the input data and saliency series are transformed into deep representations through the use of LSTM layers (Equations 6.12a and 6.12b). These deep representations are then used to learn two attention masks, on corresponding to data, the other one to saliency, through the application of the *sigmoid* operations (Equations 6.12c and 6.12d). Once the individual attention masks are learned, a mutually-supporting cross-attention mechanism is implemented through Equations 6.12e and 6.12f. As a final step of the SDMA block, the final output is determined using Equation 6.12g, which is the summation of attended data and saliency features.

6.4.4.2 Multi-Layer SDMA

In this section, *two* architectural configurations are explored to integrate SDMA attention block into a multi-layer network.

- ***Type 1***: As shown in Figure 6.10a, in the first approach a sequential architecture is constructed, where the outputs of feature and attention branches feed

to the subsequent feature and attention branch. Note that in this architecture, the saliency features are gradually abstracted along with the data features.

- ***Type 2***: In the second architecture, a semi-parallel attention structure is leveraged: as shown in Figure 6.10b, in this case the deep saliency features are learned from input saliency in each attention block opposed to deep saliency features from prior block.

This section experimentally shows that the both SDMA architectures are able to outperform the state-of-the-art. The experiments also show that the Type 2 architecture is able to provide better performance by countering the problem of over-abstraction [46].

6.5 Experiments

In this section, the SDMA framework is experimentally evaluated and compare it against alternative attention mechanism. The SDMA framework was implemented using Keras Library [22] with Tensorflow Backend [2] and feature extraction (see Table 6.1) was carried on MATLAB [59].

6.5.1 NN Architectures and Accuracy Measures

Two types of network architectures are considered, a single layer configuration and a 3-layer configuration, where the number of hidden neurons in each layer in the network is equal to twice the number of variates. All competitors have identical data branch for each competitor and custom attention block. The performance of SDMA Framework is evaluated for time series classification and forecasting. For classification tasks, "*categorical crossentropy*" and "*RMSProp*" as model loss and optimizer are used, respectively. For forecasting tasks, "*mean absolute error*" is

Table 6.1: Overview of Datasets and Parameter Configuration

Dataset	**MOCAP**	**AUSLAN**	**SML**
# Variates	62	22	17
# Timestamps	1000	45	96
# Instances	184	2585	42
# Targets	8	95	1
Task	Classification		Regression
Domain	Gesture		Energy
Epochs	2	25	50
# of octaves (O)	3	3	3
# of scales (S)	3	3	3
Initial Sigma (σ_0)	0.5	1.5	2
Intensity Threshold	0.005	0.005	0.005

reported and also use it as model loss/error. The dataset was split into training (70%), validation (10%) and testing (20%) set. Models were trained for fixed epochs (see Table 6.1) and best model performance is reported. For forecasting models $l = 1$ is used to predict target 1 step ahead in the future. Execution times are not reported, as equal number of epochs for each configuration.

6.5.2 Datasets

The performance of SDMA framework are evaluated on three benchmarks:

- **MOCAP** (Motion Capture) [1] data set was recorded at CMU's Motion Capture Lab – the subject wore an exoskeleton comprised of motion sensors recording 62 attributes for a period of 1000 time units, and for 8 gestures.

Table 6.2: Comparison of Model Performance for SDMA Framework and Various Baseline Attention Mechanisms

Evaluation Metric			**Classification Accuracy (↑)**						**Forecasting Error (↓)**		
Benchmark Datasets			**MOCAP**			**AUSLAN**			**SML**		
Architecture Type			**Single**	**Multi-Layer**		**Single**	**Multi-Layer**		**Single**	**Multi-Layer**	
			Layer	**Type 1**	**Type 2**	**Layer**	**Type 1**	**Type 2**	**Layer**	**Type 1**	**Type 2**
Base W/O Attention			72.65	68.53	68.53	71.16	69.40	69.40	0.129	0.157	0.157
Attention	**Conv. Attn.**	**Self** [71]	74.41	70.88	70.88	72.91	73.96	73.96	0.099	0.131	0.131
		DSTP [73]	N.A.	71.64	71.64	N.A.	71.05	71.05	N.A.	0.121	0.121
		SAN [33]	77.35	70.88	70.88	74.23	70.00	70.00	0.115	0.133	0.133
		FusAtNet (Data Only) [84]	74.32	74.32	74.32	81.61	81.61	81.61	0.104	0.104	0.104
	Conv. Cross-Attn	**FusAtNet (Data + Sal.)** [84]	84.12	84.12	84.12	82.15	82.15	82.15	0.089	0.089	0.089
		SDCA	75.88	82.94	81.82	80.17	78.63	82.77	0.097	0.103	0.092
	Mutual Cross-Attn	**SDMA** (*Proposed*)	**88.23**	**84.05**	**91.17**	**88.77**	**85.96**	**86.56**	**0.078**	**0.094**	**0.087**

- **AUSLAN** (Australian Sign Language) [56] datasets is comprised of 22 variates recording 95 hand gestures for 2565 instances for an average length of 45 time units.

- **SML** [81] is a data set from a building energy monitoring systems recording 24 attributes, for 40 calendar days recording attributes every 15 mins.

Table 6.1 summarizes the datasets.

6.5.3 Competitors

As competitors, five NN attention modules are considered:

- **Self Attention** [71] is a mechanism to compute relationships among input features using activation functions, such as "*sigmoid*".

- **DSTP** [73] a dual-stage two-phase attention, applies attention along variates and time in two separate stages.

Table 6.3: Performance Evaluation for Various Dataset Using SDMA Framework for Various Mask Initialization Sstrategies

Evaluation Metric			**Classification Accuracy (↑)**						**Forecasting Error (↓)**		
Benchmark Datasets			**MOCAP**			**AUSLAN**			**SML**		
Architecture Type			**Single**	**Multi-Layer**		**Single**	**Multi-Layer**		**Single**	**Multi-Layer**	
			Layer	**Type 1**	**Type 2**	**Layer**	**Type 1**	**Type 2**	**Layer**	**Type 1**	**Type 2**
Base W/O Attention			72.65	68.53	68.53	71.16	69.4	69.40	0.129	0.157	0.157
Gaussian	**Scale-Space**		83.82	77.65	85.29	73.26	74.23	83.47	0.103	0.129	0.097
	Unit	**With-Cut**	82.35	77.65	82.35	73.70	77.45	83.47	0.086	0.116	0.101
	Amplitude	**W/O-Cut**	84.12	77.94	79.12	76.04	78.42	82.32	0.087	0.112	0.104
	Center	**With-Cut**	86.18	72.65	81.18	73.89	79.35	81.72	0.085	0.125	0.103
	Amplitude	**W/O-Cut**	83.24	72.18	82.06	75.86	83.95	85.88	0.091	0.129	0.113
	Average	**With-Cut**	82.94	79.41	82.35	84.24	81.40	82.28	0.082	0.109	0.101
	Amplitude	**W/O-Cut** (*Proposed*)	**88.23**	**84.05**	**91.17**	**88.77**	**85.96**	**86.56**	**0.078**	**0.094**	**0.087**

- **SAN** [33] emulates localized event extraction within the attention block for dynamic feature extraction to adaptively drive network attention.
- **FusAtNet** [84] is a cross attention module that learns different attention masks using the two input modalities; FusAtNet is trained with the input time series and saliency series.
- **SDCA** (Saliency Driven Cross Attention) is a variant of the proposed SDMA, where only the input features are attended based on the saliency series extracted the event analysis process described in Section 6.4.

6.5.4 Results

6.5.4.1 Model Performance

Table 6.2 presents the classification accuracy results (higher the better ↑), and forecasting error results (lower the better ↓) as well. As shown in the table, models trained with SDMA as the attention blocks are able to outperform the competitors

and the SDCA variant (which does not leverage mutual-enforcement within the attention block). Existing mechanisms, such as Self and DSTP, either look at *one* single time instance along variates (fine-grained) or over the entire length of the variates (coarser), thus ignoring the localized, multi-scale, changes in various subsequences in the time series. Furthermore, SDMA is able to outperform the state-of-the-art FusAtNet (data+saliency) cross attention mechanism. It can therefore be argued that the proposed mutual cross attention mechanism in SDMA is more effective in eliminating noise. Additionally, when input series are used to learn the masks in FusAtNet (data-only), a loss in model performance can be observed; this further confirms the robustness and richness of the learned saliency series.

The table also show that a single layer SDMA architecture is often as good or better than the multi-later architectures. Among the multi-layer structures, however, the Type-2 SDMA model architecture, which leverages a semi-parallel structure to prevent over-abstraction of attention features, perform better than the Type-1 architecture.

6.5.4.2 Ablation Study

Next, the model performance is evaluated for various design choices made in the SDMA Framework through an ablation study. The results are presented in Table 6.3. In this table, an architecture without attention against the proposed mutually-enforcing saliency based attention (SDMA) architecture under different strategies for generating the underlying saliency series is considered. Here

- ***scale space*** corresponds to the version of SDMA, where the output of Eq. 6.4 is used directly to construct the per-event saliency series;

- ***unit amplitude*** corresponds to a scenario where all events are assigned the same magnitude after normalization; i.e, the output of Equation 6.9 is used as per-event saliency;
- in ***center amplitude***, the amplitude that the event center is used as the event amplitude in Eq. 6.10; and
- in ***mean amplitude***, the average of the amplitudes within the event scope is used as the event amplitude in Eq. 6.10.

The table compares the *with-cut* and *without-cut* strategies for series constructions.

As shown in the table 6.3, the saliency strategy which considers the average amplitude based scaling of saliency (which considers the entire temporal scope of the event), complemented with a *without-cut* construction (which avoids potential aberrations at the border of the event scope), consistently provides the best performing attention model.

6.6 Conclusion

This chapter presents the novel *SDMA Framework* for learning a saliency series for a given input time series. In particular, a novel approach to initialize attention as a function of localized temporal events and their temporal scopes are proposed. These events help highlight the salient sub-sequences for the network to focus on. SDMA leverages the novel mutually supporting cross-attention architecture to combine information from input series and saliency series, where "*input attending saliency*" and "*saliency attending the input*", providing superior gains then conventional and cross-attention mechanisms. Experimental evaluation on the various datasets, such as MOCAP, AUSLAN, and SML highlight that SDMA outperforms state-of-the-art attentions mechanism.

Chapter 7

XM2A: MULTI-SCALE MULTI-HEAD ATTENTION WITH CROSS-TALK

7.1 Overview

Advances in sensory technologies are enabling the capture of a diverse spectrum of real-world data streams. Increasing availability of such data, especially in the form of multi-variate time series, allow for new opportunities for applications that rely on identifying and leveraging complex temporal patterns A particular challenge such algorithms face is that complex patterns consist of multiple simpler patterns of varying scales (temporal length). While several recent works (such as multi-head attention networks) recognized the fact complex patterns need to be understood in the form of multiple simpler patterns,this chapter notes that existing works lack the ability of represent the interactions across these constituting patterns. To tackle this limitation, this chapter presents a novel *Multi-scale Multi-head Attention with Cross-Talk* (**XM2A**) framework designed to represent multi-scale patterns that make up a complex pattern by configuring each attention head to learn a pattern at a particular scale and accounting for the co-existence of patterns at multiple scales through a cross-talking mechanism among the heads. Experiments show that XM2A outperforms state-of-the-art attention mechanisms on benchmark datasets.

7.2 Introduction

Recent advances in sensor technologies have enabled assimilation of large amount of streaming data in a wide variety of applications, from gesture recognition [34] to object recognition [113] and speech processing [85]. In particular,

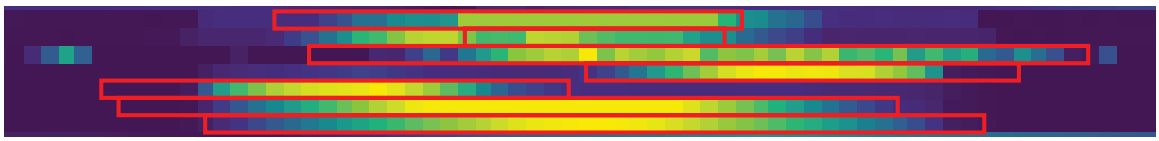


Figure 7.1: Overview of Salient Multi-Scale Temporal Patterns Extracted from a Multi-Variate Time Series [96]

sensor networks have enabled simultaneous recording of a multitude of attributes, leading to large multi-variate time series - each variate corresponding to a different attribute being recorded. This explosion in the pace of multi-variate temporal data generation has highlighted the need for effective time series analytics, helping discover complex patterns captured in temporal data.

7.2.1 Multi-Scale Feature Learning

Localized multi-scale features extraction approaches, such as SIFT [76] and RMT [15, 75] (see Figure 7.1), have shown promise at identifying salient and robust information in image matching and time series analytics, respectively. In a similar vain, neural networks learn multi-scale features by interleaving trainable layers (capturing different aspects of the data) and pooling layers (to vary the scale of feature maps) [106, 110]. [128] has shown that combining salient local patterns learned by [76] and global feature maps discovered through a deep neural network can improve model performance in image analysis. [33] has shown that the model performance can be further boosted by simulating the localized multi-scale feature extraction process within the network itself when learning attention masks.

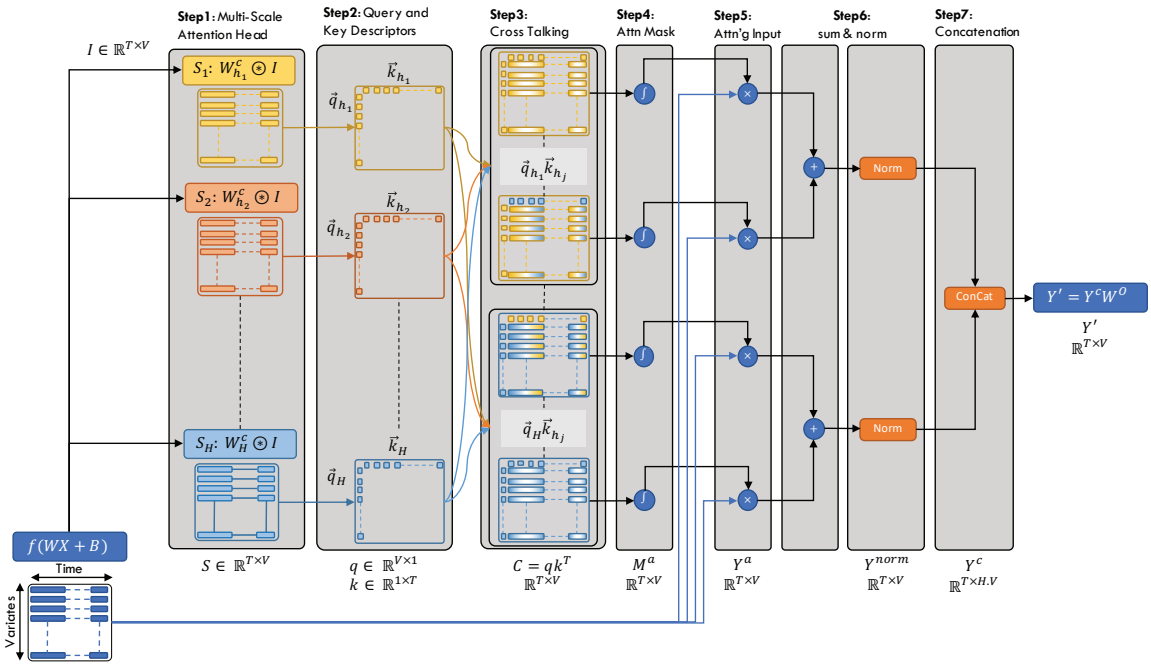


Figure 7.2: An Abstract Overview of the Proposed XM2A Framework, Introducing the Cross-Talk Between Attention Heads to Share Multi-Scale Information Learned on Each Head Independently; This is Followed by Learning a Rich Attention Mask Capturing Information from Multi-Head. i.e. Information Learned Using Kernel of Different Size at Each Attention Head

7.2.2 Time Series Analysis with Multi-Head Attention

Recurrent networks and LSTMs [99, 48, 20], where the goal is to learn a predictor, $p : \mathbb{T}|\boldsymbol{W} \rightarrow Y$ to map the input ($\mathbb{T}$) to the output (Y) using model parameter ($\boldsymbol{W}$), may require large amounts of resources due to their computational complexities that slow the model training and can reduce the model effectiveness. Furthermore, real-world time series, often contain noisy information that can impact the model performance. [114] pointed that if the input (or latent) features are properly attended, one can eliminate the need for recurrent units [99, 48, 20]. This not only

enables to use of computationally efficient *dense* and *convolutional* units, but also enables robustness of the model against noise in the input data. [114] further proposed that it may be possible to capture diverse relationships between a pair of timestamps by using multiple operations, called "*attention heads*", simultaneously. While this approach has shown promise in neural machine translation tasks, unfortunately it fails to account for the fact that local temporal patterns of different scales can be important, moreover, the proposed operations do not account for patterns spanning multiple timestamps. [40], recently, proposed a multi-scale multi-head self-attention (MSMSA) mechanism to learn multi-scale features with the attention heads to capture local patterns at different scales. However, the design of the multi-scale module in [40] does not account for the interactions among the attention elements of different scales.

7.2.3 Key Contributions

As outlined above, multi-variate time series often include robust localized patterns, *in the form of sub-sequences*, that carry information critical to a given target variable, $\boldsymbol{Y}$, and attention mechanisms leverage these patterns to focus the computation to relevant parts of the data. Conventional approaches, such as [79, 7, 114], only consider sub-sequences of fixed length, whereas recent multi-scale work, such as [40], fail to account for the interactions among the patterns of different scales. To tackle this limitation, this chapter presents the proposed novel *Multi-scale Multi-head Attention with Cross Talk* (**XM2A**) framework designed to represent multi-scale patterns that make up a complex pattern by configuring each attention head to learn patterns at a particular scale and accounting for co-existence of patterns at multiple scales through a cross-talking mechanism among the attention heads (Figure 7.2):

- **Multi-Scale, Multi-Head Attention with Small Number of Trainable Parameters**: Traditional multi-head attention modules learn patterns at a single scale (see Section 7.4.1). XM2A argue that this is a critically flawed assumption that does not hold true for many applications; in contrast, these applications involve a spectrum patterns at different lengths. As opposed to conventional attentional mechanisms that learn the query and key-value pair directly by applying a linear transformation using weight matrices, XM2A uses convolutional transformations in each attention head to help capture patterns at different temporal scales. Moreover, in contrast to the existing work, such as [40], the proposed XM2A attention framework introduces a smaller number of trainable patterns, significantly improving efficiency and effectiveness of the learning process.

- **Attention Heads with Cross Talk**: In XM2A, different attention heads learn features of different scales that co-exist in the time series. However, it must be noted that naïve concatenation of attention outputs may not necessarily be the most effective way to capture the interactions among the multi-scale features. Therefore, XM2A proposes a cross-talking mechanism among the attention heads (Figure 7.3): a feature descriptor summarizes temporal and variate level information at each attention head and the cross-talk among attention heads allow the attention processes at multiple scales to share information to help learn a strong predictor.

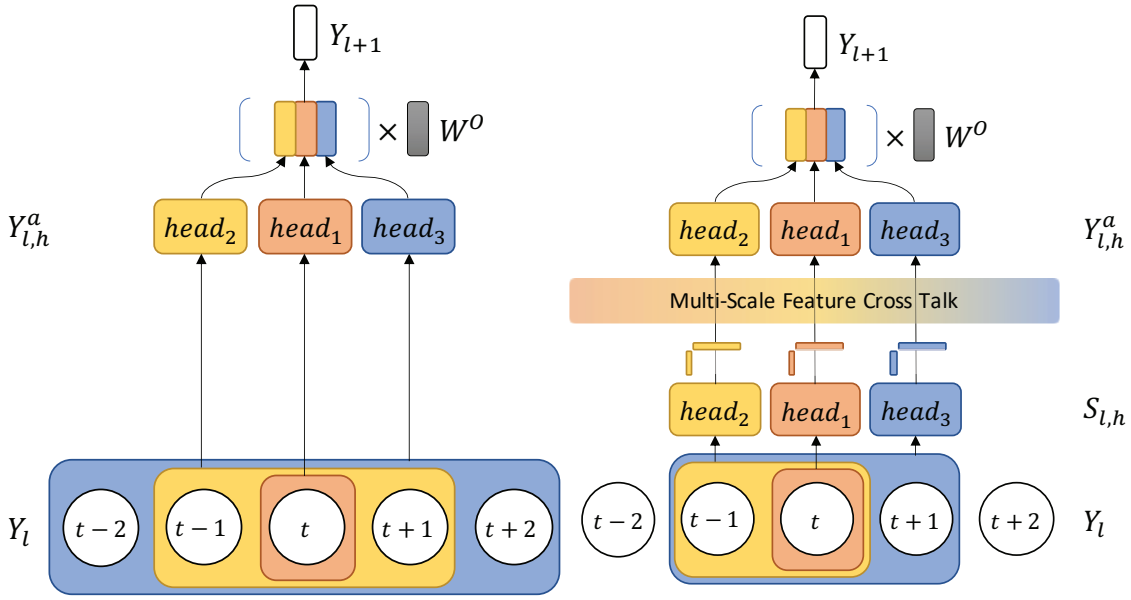
Experiments reported in Section 7.6 show that the proposed XM2A mechanism outperforms state-of-the-art attention mechanisms, such as Transformers and MSMSA, on benchmark datasets, such as SADD, AUSLAN, and MOCAP.

7.3 Related Work

Multi-scale feature learning has long attracted the interest of pattern recognition community, from localized feature extraction [76, 15, 75] or deep feature [106, 110] learning multimedia analytical tasks. Deep features have shown promising results in capturing features maps at different scales by stacking multiple layers and have shown human-level performance [106, 110, 114]. Their extension to the domain of time series, using RNNs [99], LSTMs [48], or GRUs [20], has shown promising results as well. However, recurrent units are inherently complex and significantly increase the model training time.

[7] observed that while the complexity of the recurrent units cannot be reduced, an informed attention mechanism can help the models intelligently select a subset of informative input features to improve the model performance. Transformer [114] further observed that, when attention mechanism is used wisely, it can completely eliminate the need of recurrent units in the network and one can replace them with dense or convolutional unit. [114] proposed to use multiple attention head simultaneously in order to learn the relationships between different timestamps in the time series (or tokens in the sentence embedding). However, transformer lacks the ability to consider localized temporal patterns at different scales in a given multi-head attention block. [40] proposed to learn multi-scale features of different scales specific to each attention head, where the feature scale was determined as $(2 * h - 1)$ where $1 \leq h \leq H$ is the attention head identifier. However, [40] fails to observe that these multi-scale features co-exists in the original time series and that these features can interact. Furthermore, the heuristic of [40] leads to a significant increase in the number of trainable parameters. Based on these observations, XM2A framework aims to discover multi-scale features with cross-talking attention

heads to learn rich attention masks.



(a) MSMSA [40]

(b) XM2A (*proposed*)

Figure 7.3: XM2A vs. MSMSA [40]

7.4 XM2A Framework

A uni-variate time series (UVTS), $\boldsymbol{T}$, is a sequence of ordered pair of observations and times at which observations were recorded for a given attribute (variate); i.e. $\boldsymbol{T} = [(\mathsf{v}_1, \mathsf{t}_1), (\mathsf{v}_2, \mathsf{t}_2), \ldots, (\mathsf{v}_T, \mathsf{t}_T)]$. A multi-variate time series (MVTS), $\mathbb{T} \in \mathbb{R}^{V \times T}$, is a set of UVTS, $\boldsymbol{T} \in \mathbb{R}^{1 \times T}$, s.t. $\mathbb{T} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots, \boldsymbol{T}_V\}$ where V and T are the numbers of variates and timestamps, respectively. Each MVTS is associated with a symbolic label Y. XM2A represents this as a probability distribution where true label has a probability of 1 and false labels 0. In time series classification, the goal is to learn a predictor, $p : \mathbb{T}|\boldsymbol{W} \rightarrow Y$ that maps the input ($\mathbb{T}$) to the output (Y) given model parameters ($\boldsymbol{W}$) in a way that minimizes the separation between the true output Y and predicted output $Y' = p(\mathbb{T}|\boldsymbol{W})$, i.e. $\triangle(Y, Y')$ – more

details on this in Section 7.6.3.

7.4.1 Conventional Multi-Head Attention

Conventional attention mechanisms rely on the immediate past to learn importance of the data [51]; they also learn a single mapping (relationship) between the query ($\boldsymbol{Q}$ – a given timestamp) and the keys ($\boldsymbol{K}$ – all timestamps) [79]. [114] proposed an attention function which, for a given input, $\boldsymbol{I}$, maps a query, $\boldsymbol{Q}$, and a set of key-value pairs, $\langle \boldsymbol{K}, \boldsymbol{V} \rangle$, to an output, $\boldsymbol{O}$, computed as a weighted sum of the input values. Here, the query, $\boldsymbol{Q}$, the key, $\boldsymbol{K}$, and the values, $\boldsymbol{V}$, are all obtained through linear transformations on the input, $\boldsymbol{I}$ (i.e., $\boldsymbol{Q} = \boldsymbol{W}^Q \boldsymbol{I}$, $\boldsymbol{K} = \boldsymbol{W}^K \boldsymbol{I}$, and $\boldsymbol{V} = \boldsymbol{W}^V \boldsymbol{I}$). The output, $\boldsymbol{O}$, is computed as

$$\boldsymbol{O} = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_k}}). \tag{7.1}$$

Here, $\sqrt{d_k}$ is a scaling factor defined in terms of the dimensionality of the key vector. This scaling factor is used for preventing the intermediate features from exploding due to the magnitude of the query and key matrices. The attention head is defined as a scaled dot product:

$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_k}})\boldsymbol{V}. \tag{7.2}$$

By extension, a multi-head attention block is defined as

$$MultiHead(\boldsymbol{I}, h) = concat(head_1, \ldots, head_H)\boldsymbol{W}^O, \tag{7.3}$$

where $1 \leq h \leq H$ denotes one of the H heads in the model, $head_h = Attention(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h)$. Finally, the output weight matrix $\boldsymbol{W}^O \in \mathbb{R}^{H.d_k \times V}$ is used to map the output of the concatenation back to the original number of variates.

7.4.2 Multi-Scale Attention Heads in XM2A

Multi-head attention process enables the network to simultaneously learn multiple relationships among the query and the keys in the series. While multi-head attention has shown promise in a variety of neural machine translation tasks, this chapter observes that the conventional multi-head attention has significant limitations – in particular, when attending many real-world time series, it is important to look at local temporal neighborhoods of the query as patterns that may span multiple scales. Leveraging multi-scale patterns can reduce the amount of noisy information being propagated in the network, as different scales perform smoothing at different levels; thus suppressing the details differently at different scales.

Based on these observations, XM2A uses convolutional transformations, similar to MSMSA, in each attention head to help capture patterns at different temporal scales, as opposed to conventional attentional mechanisms that learn the query and key-value pair directly by applying a linear transformation using weight matrices ($\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V$).

7.4.2.1 Learning Multi-Scale Patterns

To learn multi-scale temporal patterns, XM2A transforms the linear operation of learning query, key, and value from a given input ($\boldsymbol{I}$), into a $1D$ convolutional operation on a given attention head (h), i.e.

$$\boldsymbol{W}_h^{\dagger}\boldsymbol{I} \rightarrow \boldsymbol{W}_h^{c} \circledast \boldsymbol{I}.$$

Here, † represents the individual weight matrix for query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$) and value ($\boldsymbol{V}$), and $\boldsymbol{W}^c \in \mathbb{R}^{d_k \times s \times 1}$ is the weight matrix for convolution operation ($\circledast$) where d_k represents the number of convolution kernels, and $s \times 1$ is the size of the kernel –

i.e. the scale of temporal patterns learned. Given these, multi-scale patterns, $\boldsymbol{S}$, learned by the process (Step 1 in Figure 7.2) can be formally defined as,

$$\boldsymbol{S}_h = \boldsymbol{W}_h^c \circledast \boldsymbol{I}. \tag{7.4}$$

Note that, in XM2A, the scale of the temporal patterns learned is a function of the number of attention heads: an attention module with H heads will learn temporal patterns with scales $s \in [1, \ldots, H]$. More specially, $head_1$ will learn temporal patterns of length 1, $head_2$ will capture patterns of length 2, and so on.

7.4.2.2 Query and Key Descriptors

Unlike conventional attention processes, including MSMSA, which simply use scaled dot-product between the query and key matrix, XM2A extracts and leverage *query* and *key descriptors*, both along the time and variate dimensions, to re-calibrate the intermediate network output; i.e. to learn the importance of each element in the value matrix $\boldsymbol{V}$. The key intuition behind the use of descriptors in the attention process is that

there exists inherent interdependencies among the variates and, similarly, among the timestamps and these interdependencies can be extracted and represented in the form of query and key descriptors.

as shown in Step 2 Figure 7.2):

- **Query descriptors** ($\vec{q} \in \mathbb{R}^{V \times 1}$) can be interpreted as queries learned through global averaging of the multi-scale patterns along the temporal dimension. They are extracted by compressing the information captured for each variate

into a single value, s.t.

$$\forall v = 1, \ldots, d_k \;\; \vec{q}_h[v] = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{S}_h[t, v]. \tag{7.5}$$

- **Key descriptors** ($\vec{k} \in \mathbb{R}^{T \times 1}$) can be interpreted as keys learned through global averaging of the multi-scale patterns along the temporal dimension. They are extracted by compressing the information captured for each timestamp into a single value, s.t.

$$\forall t = 1, \ldots, T \;\; \vec{k}_h[t] = \frac{1}{d_k} \sum_{v=1}^{d_k} \boldsymbol{S}_h[t, v]. \tag{7.6}$$

The *average* operation in Equation 7.6 and 7.5 helps suppress noise in the intermediate data. While the *maximum* operation can also be instead of *average*, XM2A observes that *maximum* operation would leave the network susceptible to outliers. In Section 7.6.4.3, experimentally show that models learned using *average* based descriptors out-perform models learned using *maximum* based descriptors.

7.4.3 Cross-Talking among Attention Heads

In conventional multi-head attention networks, each attention head independently learns an attention matrix without implicit or explicit knowledge of the learnings at other attention heads. This is referred as *self-talking* attention heads; the proposed key (temporal) and query (variate) descriptors can be used for learning an attention matrix, $\boldsymbol{M}_h^a \in \mathbb{R}^{T \times V}$ (recording the attention weight for each element in the value matrix) by performing a dot-product between the two descriptors, as follows:

$$\boldsymbol{M}_h^a = softmax(\vec{q}_h \vec{k}_h^{\mathsf{T}}). \tag{7.7}$$

Self-talking approach for learning the attention matrix does not capture the interactions among the multi-scale attention heads. Therefore, XM2A establishes a cross-talking mechanism among the attention heads to facilitate information sharing among the heads.

Firstly, instead of learning a *single* attention matrix per attention head, XM2A computes one cross-talking attention matrix per attention head pair (Steps 3-4 Figure 7.2) as follows:

$$\forall_{1\leq h_1,h_2\leq H} \;\; \boldsymbol{M}^a_{h_1,h_2} = softmax(\vec{q}_{h_1}\vec{k}^{\intercal}_{h_2}) \tag{7.8}$$

Once the cross-talking attention matrices are computed, the value matrix is attended by each mask (Step 5 Figure 7.2):

$$\forall_{1\leq h_1,h_2\leq H} \;\; \boldsymbol{Y}^a_{h_1,h_2} = \boldsymbol{M}^a_{h_1,h_2} \odot \boldsymbol{V}_{h_1}. \tag{7.9}$$

Next, XM2A applies an "*sum*&*norm*" operation on the outputs of each individual attention; i.e. each attention head returns only a single output (Step 6 Figure 7.2):

$$\forall_{1\leq h_1\leq H} \;\; \boldsymbol{Y}^{norm}_{h_1} = norm(\sum_{h_2=1}^{H} \boldsymbol{Y}^a_{h_1,h_2}). \tag{7.10}$$

Here, the summation allows for merging different temporal patterns captured at different scales. The *normalization* step that follows help prevent the gradients from exploding due to potentially large sums.

In the final step, XM2A concatenates the individual attention head outputs and computes a final output, $\boldsymbol{Y}'$, of the multi-scale, multi-headed attention module as a whole:

$$\boldsymbol{Y}' = concat(\boldsymbol{Y}^{norm}_1, \ldots, \boldsymbol{Y}^{norm}_H)\boldsymbol{W}^O. \tag{7.11}$$

The use of the output weight matrix, $\boldsymbol{W}^O$ (as shown in Step 7 Figure 7.2), helps fuse the information learned across different multi-scale attentions into a single

Table 7.1: Feature Scale and # of Parameters for Different XM2A Variants

Variant	**Length of h^{th} Feature**	# of Parameters
MSMSA	$(2 * h) - 1$	$(2 * h) - 1$
XM2A-MSMSA (M)	$(2 * h) - 1$	$(2 * h) - 1$
XM2A-Exp (E)	2^{h-1}	2^{h-1}
XM2A-Exp-Pool (EP)	2^{h-1}	1
XM2A-Pool (P)	h	1
XM2A-Hybrid (H)	$\lceil \frac{2^{h-1}}{h} \rceil$	h
XM2A	h	h

output of the multi-scale multi-head attention block, as the multi-scale patterns co-exists in the input series ($\mathbb{T}$).

7.5 Versions of XM2A

In the default XM2A formulation presented in the previous section, the H heads have been constructed by incrementing the attention length by one for each head starting from 1; i.e., $1 \leq h \leq H$. However, XM2A sees that different versions of XM2A are possible depending on how the attention lengths have been varied and how many kernel are allocated per attention head:

- XM2A-MSMSA (M): This is the version of the XM2A algorithm, where the feature temporal length and the number of parameters in a kernel is determined as $2 * h - 1$
- XM2A-Exp (E): This is the version of the XM2A algorithm, where the feature temporal length and the number of parameters in a kernel is determined as 2^{h-1} to cover wider spread of information

Table 7.2: Overview of Datasets and Parameter Configuration

Dataset	SADD	AUSLAN	MOCAP
# Instances	8800	2585	184
# Timestamps	50	45	1000
# Variates	13	22	62
# Targets	10	95	8
Training Epochs	25	25	5

- XM2A-Pool (P): This version applies the pooling operation of size h, and the feature temporal length is set as 1 to cover identical of information to XM2A without increase in number of parameters
- XM2A-Exp-Pool (EP): This version applies the pooling operation of size 2^{h-1} and feature scale of 1 to cover wider (than XM2A) spread of information without increase in number of parameters

Model accuracy and the impact of parameter complexity are evalauted for different versions of XM2A in the next section.

7.6 Experiments

In this section, the XM2A framework is evaluated and compare it against alternative attention mechanisms. XM2A was implemented using Keras Library [22] with Tensorflow [2].

7.6.1 Datasets and Network

Three multi-variate time series benchmark datasets are considered:

- **SADD** [41] contains 8800 spoken instance of digits ($0 \ldots 9$) in arabic and rep-

resented using 13 MFCCs over 50 time units (*mel frequency cepstral coefficient*).

- **AUSLAN** [56] is comprised of 22 variates recording 95 hand gestures for 2565 instances for an average length of 45 time units.

- **MOCAP** [1] consists of full-body gestures performed by *8* subjects recorded using exoskeleton capturing 62 attributed over 1000 time units.

Datasets are summarized in Table 7.2. Each series $\boldsymbol{T}$ in the multi-time series $\mathbb{T}$ (see Section 7.4) is transformed to have mean of *zero* and *unit* standard deviation. This transformation has been shown to improve the model convergence during training [66, 117, 55, 6].

7.6.2 Competitors

XM2A was compared against various non-attentioned and attentioned (single-head and multi-head) networks as follows:

- **Baseline networks**: VGG [106] and InceptionNet [110] were considered as baselines, as VGG learns multi-scale features with the increase and network depth and InceptionNet simultaneously learn features of varying scales within a layer.

- **Self Attention**: [79] proposes to let the data attend itself, i.e. $softmax(\vec{q}_t^{\intercal}.\vec{k}_t)\vec{v}_t$, where, q, k, and v are not linearly transformed.

- **SENet**: [51] proposes to calibrate each neuron output with an identical score by squeezing the entire neuron ($\boldsymbol{Y}_{n_i}$) output into a single value, $s_i = avg(\boldsymbol{Y}_{n_i})$ and then passing it through an autoencoder to learn latent relationships.

- **BAM**: [91] creates *three* branches: 1) data, 2) temporal, and 3) variate branches. The temporal and variate branches, learn the attention mask exclusively for time and variate information respectively and, in the data branch, the two masks are fused together to attend the input data.

- **Transformer**: [114] proposes to parameterize [79] and to use multiple attention heads simultaneously. An attention head is defined as follows:

$$head(\vec{q}\boldsymbol{W}^{Q}, \vec{k}\boldsymbol{W}^{K}, \vec{v}\boldsymbol{W}^{V});$$

  the multi-head attention block is defined as a function of multiple heads:

$$MultiHead(\vec{q}, \vec{k}, \vec{v}) = [head_1, \ldots, head_H]\boldsymbol{W}^{O}.$$

- **MSMSA**: [40] transforms each attention head from [114] to learn multi-scale patterns, without cross-talk, where the scale is determined by $2 \times h - 1$; here h is the attention head identifier.

7.6.3 Model Training and Configuration

Models were trained for time series classification tasks. For model training, "*categorical cross-entropy*" and "*RMSProp*" were used as model loss and optimizer respectively. Each dataset was split into training (70%), validation (10%) and testing (20%) sets. As defined in [114], the model contained 8 attention heads, 64 hidden neusons in each attention head, and 6 layers. In the experiments best model accuracy is reported after training the model for 25 epochs.

Table 7.3: Comparison of Model Classification Accuracy of XM2A Against state-of-the-art Attention Networks

Model Configuration			**Classification Accuracy**		
			SADD	**AUSLAN**	**MOCAP**
Baseline W/O Attn		**Deep Net** [106]	95.93	80.50	71.47
		Wide Net [110]	96.01	81.29	72.29
Attention	**Single-Head**	**Self** [79]	94.56	82.50	73.76
		SENet [51]	94.91	82.54	73.11
		BAM [91]	95.95	82.89	74.99
	Multi-Head	**Transformer** [114]	96.16	85.52	75.11
		MSMSA [40]	97.03	87.63	76.47
		XM2A	**98.12**	**89.52**	**78.82**

7.6.4 Results

7.6.4.1 Parameter-Accuracy Tradeoff

In Figure 7.4 presents the parameter-accuracy tradeoff analysis for the various variants of XM2A, along with the basic Transformer and MSMSA architectures. As it can be seen in the figure, MSMSA provides better accuracy than Transformer, but requires a significantly higher number of parameters. In contrast, XM2A variants always provide better accuracy than both Transformer and MSMSA and the number of parameters can be as low as that of Transformer (for XM2A-P). The default XM2A provides the best overall accuracy and the default version is considered in the rest of the section.

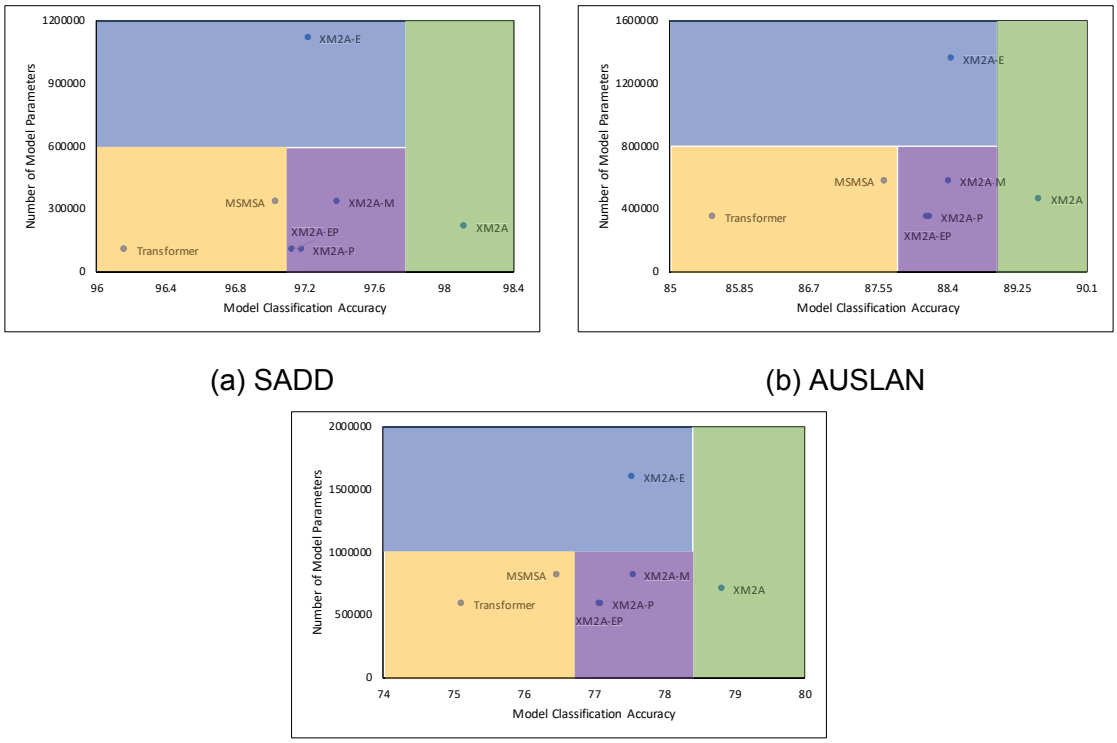
Table 7.4: Model Accuracy for Various Model Configurations

Model Configuration			**Classification Accuracy**		
Feature Type	**Data Norm.**	**Cross Talking**	**SADD**	**AUSLAN**	**MOCAP**
Maximum	**–**	**ST**	94.99	80.40	71.76
	Batch	**IS**	96.02	85.70	74.94
		SS	96.24	86.24	76.89
	Layer	**IS**	95.96	86.12	75.41
		SS	97.52	88.15	77.99
Average	**–**	**ST**	95.12	80.73	72.17
	Batch	**IS**	96.21	86.56	76.14
		SS	96.02	87.38	77.05
	Layer	**IS**	96.50	86.93	76.47
		SS (XM2A)	**98.12**	**89.52**	**78.82**

7.6.4.2 Model Performance against Competitors

Table 7.3 presents the classification accuracy results for XM2A and compare it against various state of the art attention mechanisms. As it can be seen in the table, models trained with XM2A-based multi-head attention module outperforms not only the traditional attention modules, such as Self, SENet[1], and BAM, but also, the Transformer, and MSMSA. It is important to note that, XM2A leads to signif-

[1]While not critical to this discussion, it is interesting to note that Self and SENet based attention mechanisms lead to a drop in model performance for the SADD dataset – this is due to the presence of ReLU activation in the attention block, which eliminates the negative MFCC signals that are of significant importance in the domain of speech recognition.



(a) SADD

(b) AUSLAN

(c) MOCAP

Figure 7.4: Trade-off Between Number of Model Parameters and Model Accuracies

icant gains in model performance against MSMSA. This confirms the argument that that there co-exist local patterns of different scales and they, together, define a complex pattern to attend – therefore, allowing simpler multi-scale patterns learned at different attention heads share their knowledge through cross talking help discover complex patterns that provide superior attention performance. The fact that XM2A requires a smaller number of model parameters than MSMSA (as seen in Figure 7.4) further confirms the significance of cross-talking between the attention heads.

7.6.4.3 Ablation Study

This section presents the evaluation of various design choices made in the XM2A framework. In particular, the key and query descriptor generation (Eq 7.5 & 7.6), cross-talking (Eq 7.8) and normalization steps (Eq 7.10) are evaluated: **(a)** *Descriptors* can be generated in two ways: *averaged* and *maximum* descriptors. **(b)** Two different normalization strategies are considered: Batch [55] and Layer Norm [6]. **(c)** Further different talking mechanisms were considered to generate attention masks: in *self-talk* (ST), (Eq 7.7) uses the key and query descriptors within each attention head exclusively (i.e. no cross-talk); in *input-to-scale* (IS), cross-talking generates key and query descriptors for V (the value matrix) and cross talk is between the query of input and key of the head and key of the value and query of the head, respectively; *scale-to-scale* (SS), as defined in Eq 7.9, is the proposed *scale-to-scale* cross-talk mechanism for XM2A.

As it can be seen in the Table 7.4, XM2A outperforms all alternative configurations. In general, a better model performance is observed with averaged descriptor. This is because average operation is less susceptible to outliers (noisy activations) than maximum operation. Moreover, it can be seen that the layer normalization leads to a superior performance as it preserves inter-neuron relations in a layer as opposed to batch norm, which treats each neuron activation independently. Finally, the proposed multi-scale cross-talk mechanisms provides performance boosts against self-talk (or no crosstalk) and input-to-scale cross-talking strategies.

7.7 Conclusion

In this work, it is observed that existing multi-head attention techniques lack the ability of capture multi-scale patterns simultaneously in the attention module and fail to account for the co-existence of such patterns at different scales. Therefore, a novel Multi-scale Multi-head Attention with Cross-Talk, **XM2A Framework** is proposed that can capture multi-scale patterns by enabling attention heads to learn patterns at a particular scale, while accounting for the co-existence of other patterns at other scales through a cross-talking mechanism among the attention heads. XM2A outperforms state-of-the-art attention mechanisms, such as Transformers and MSMSA, on benchmark datasets, SADD, AUSLAN, and MOCAP.

Chapter 8

CONCLUSION AND FUTURE WORK

The desiderata of this dissertation are to design data-driven approaches that can help learn a high-performing deep neural network. The works present in this dissertation focus on leveraging the key insights captured in data, in particular, input data, hidden (intermediate) data, and output (model inferences) data. To this end, multiple works are proposed in this dissertation that leveraging of the data analysis techniques to learn accurate models.

8.1 Robust Allocation of Convolution Kernel

A pre-training analysis of data can uncover insights in the data in form of local multi-scale features that are of various sizes and complexities, and have diversity and distributions. Chapter 3 presents the RACKNet framework that can minimize the need for hand-crafting, by relying on the pre-training analysis of input data itself. To achieve this, RACKNet presents several hypotheses that link the properties of the localized image features to the neural networks and then, relying on these hypotheses, RACKNet framework aims to learn multiple hyper-parameters by extracting information encoded in the input datasets.

8.2 Output Information Sparsification

Complex datasets rely on a large number of network parameters for learning accurate models, however, this often introduces noisy and irrelevant model parameters that have insignificant contribution to the model output. Chapter 4 presents the iSparse framework that can sparsify the network parameters in a pre-trained

network without impacting the network performance. iSparse leverages a novel edge significance score to determine the importance of an edge concerning the final network output.

8.3 Scale-Space Attention Network

Each layer in the network learns features of a particular scale, and as the network gets deeper the sclae of the feature increases. However, while attending a given layer in the network, the attention mechanism can only leveraging feature of a particular scale. Chapter 5 presents the SAN framework that leverage output of two adjacent layers in the network to compare the changes in the activation across multiple scale through the user of novel difference-of-convolutions, followed by the feature augmentation to highlight the saliency local extrema in the data. This work highlights the fact that understanding the chances happening across the layers in the network can help boost the model accuracy.

8.4 Saliency-driven Mutual Cross Attention

Additional pre-training analysis of the data can help separate noisy irrelevant information from the relevant information. Chapter 6 presents the SDMA framework that specializes in highlighting the relevant information in the data while suppressing the noisy irrelevant information through the extraction of an additional input modality for the network called saliency series. This saliency series representation salient points in the data and their local neighborhood that is of prominence.

8.5 Multi-Scale Multi-Head Attention with Cross Talk

This dissertation observes that simpler multi-scale features co-exist in the data and together summarize the complex patterns. However, conventional attention

mechanisms fail at leveraging the co-existence of multi-scale features to describe the complex patterns. Chapter 7 presents the XM2A framework, that explicitly learns multi-scale features in each attention heads and shares the knowledge between each attention head with the mechanism called cross talk. This knowledge sharing allows for rich attention mask extraction.

8.6 Future Work

Multi-scale features have shown promise in both deep networks and conventional machine learning. With the rise in real-world applications, complex patterns, especially temporal patterns, can span both in time and across multiple attributes. This dissertation primarily explores patterns in the individual attributes (variate) without incorporating the inter-variate relationship. As future work, the works in this dissertation, especially, SDMA and XM2A can be explored further with the incorporation of the graph convolutional networks as an additional input modality to supply the network with more information to help model the relationships better.

BIBLIOGRAPHY

[1] “Cmu graphics lab motion capture database”, (2015).

[2] Abadi, M., P. Barham and et. al., “Tensorflow: A system for large-scale machine learning”, in “USENIX OSDI”, (2016).

[3] Abavisani, M., L. Wu, S. Hu, J. Tetreault and A. Jaimes, “Multimodal categorization of crisis events in social media”, in “Computer Vision and Pattern Recognition (CVPR)”, (2020).

[4] Alom, M. Z., T. Josue, M. N. Rahman, W. Mitchell, C. Yakopcic and T. M. Taha, “Deep versus wide convolutional neural networks for object recognition on neuromorphic system”, arXiv preprint arXiv:1802.02608 (2018).

[5] Ashouri, A. H., T. S. Abdelrahman and A. D. Remedios, “Fast on-the-fly retraining-free sparsification of convolutional neural networks”, (2018).

[6] Ba, J. L., J. R. Kiros and G. E. Hinton, “Layer normalization”, arXiv preprint arXiv:1607.06450 (2016).

[7] Bahdanau, D., K. Cho and Y. Bengio, “Neural machine translation by jointly learning to align and translate”, in “International Conference on Learning Representations (ICLR)”, (2015).

[8] Bay, H., A. Ess, T. Tuytelaars and L. Van Gool, “Speeded-up robust features (surf)”, Computer Vision and Image Understanding (CVIU) (2008).

[9] Bay, H., T. Tuytelaars and L. Van Gool, “Surf: Speeded up robust features”, in “European conference on computer vision”, pp. 404–417 (Springer, 2006).

[10] Bergstra, J. and Y. Bengio, “Random search for hyper-parameter optimization”, Journal of Machine Learning Research (JMLR) **13**, Feb, 281–305 (2012).

[11] Bergstra, J. S., R. Bardenet, Y. Bengio and B. Kégl, “Algorithms for hyper-parameter optimization”, in “Advances in neural information processing systems”, pp. 2546–2554 (2011).

[12] Blei, D. M. and J. D. Lafferty, “Dynamic topic models”, in “International Conference on Machine Learning (ICML)”, (2006).

[13] Bonacich, P., “Power and centrality: A family of measures”, American Journal of Sociology **92**, 5, 1170–1182 (1987).

[14] Bridle, J. S., “Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters”, in “NIPS”, (1990).

[15] Candan, K. S., R. Rossini, M. L. Sapino and X. Wang, “sdtw: Computing dtw distances using locally relevant constraints based on salient feature alignments”, Very Large Databases (VLDB) **5**, 11 (2012).

[16] Ceroni, A., C. Ma and R. Ewerth, “Mining exoticism from visual content with fusion-based deep neural networks”, in “Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval”, pp. 37–45 (ACM, 2018).

[17] Chandakkar, P. S. and B. Li, “A structured approach to predicting image enhancement parameters”, in “Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on”, pp. 1–9 (IEEE, 2016).

[18] Chen, T., H. Yin, H. Chen, R. Yan, Q. V. H. Nguyen and X. Li, “Air: Attentional intention-aware recommender systems”, in “International Conference on Data Engineering (ICDE)”, (IEEE, 2019).

[19] Chen, W., J. Wilson, S. Tyree, K. Weinberger and Y. Chen, “Compressing neural networks with the hashing trick”, in “International Conference on Machine Learning (ICML)”, pp. 2285–2294 (2015).

[20] Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, Conference on Empirical Methods on Natural Language Processing (2014).

[21] Choi, Y., M. El-Khamy and J. Lee, “Universal deep neural network compression”, Journal of Selected Topics in Signal Processing (2020).

[22] Chollet, F. *et al.*, “Keras”, (2015).

[23] Collobert, R., J. Weston and et. al., “Natural language processing (almost) from scratch”, Journal on Machine Learning Research (JMLR) (2011).

[24] Dai, J., M. Zhang, G. Chen, J. Fan, K. Y. Ngiam and B. C. Ooi, “Fine-grained concept linking using neural networks in healthcare”, in “Special Interest Group on Management of Data (SIGMOD)”, (2018).

[25] Dalal, N. and B. Triggs, “Histograms of oriented gradients for human detection”, in “2005 IEEE computer society conference on computer vision and pattern recognition (CVPR’05)”, vol. 1, pp. 886–893 (IEEE, 2005).

[26] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 248–255 (IEEE, 2009).

[27] Denil, M., B. Shakibi, L. Dinh, M. Ranzato and N. De Freitas, “Predicting parameters in deep learning”, in “Advances in neural information processing systems (NIPS)”, pp. 2148–2156 (2013).

[28] Ester, M., H.-P. Kriegel, J. Sander and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise”, in “KDD”, (AAAI Press, 1996).

[29] Felzenszwalb, P. F., R. B. Girshick, D. McAllester and D. Ramanan, “Object with discriminatively trained part-based models”, IEEE Transaction on Pattern Analysis and Machine Intelligence (TPAMI) (2010).

[30] Frankle, J. and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks”, International Conference on Learning Representations (ICLR) (2019).

[31] Garg, Y., *Multi-Variate Time Series Similarity Measures and Their Robustness Against Temporal Asynchrony* (ASU, 2015).

[32] Garg, Y. and K. S. Candan, “Racknet: Robust allocation of convolutional kernels in neural networks for image classification”, in “International Conference on Multimedia Retrieval (ICMR)”, pp. 315–323 (2019).

[33] Garg, Y., K. S. Candan and M.-L. Sapino, “San: Scale-space attention network”, in “International Conference on Data Engineering (ICDE)”, (2020).

[34] Garg, Y. and S. R. Poccia, “On the effectiveness of distance measures for similarity search in multi-variate sensory data”, in “International Conference on Multimedia Retrieval (ICMR)”, (2017).

[35] Glorot, X. and et. al., “Deep sparse rectifier neural networks”, in “International Conference on Artificial Intelligence and Statistics (ICAIS)”, (2011).

[36] Goodfellow, I., Y. Bengio, A. Courville and Y. Bengio, *Deep learning*, vol. 1 (MIT press Cambridge, 2016).

[37] Goodfellow, I., D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, “Maxout networks”, in “International Conference on Machine Learning (ICML)”, (2013).

[38] Gross, R. and J. Shi, “The cmu motion of body (mobo) database”, (2001).

[39] Grossberg, S., “Nonlinear neural networks: Principles, mechanisms, and architectures”, Neural Networks (NN) (1988).

[40] Guo, Q., X. Qiu, P. Liu, X. Xue and Z. Zhang, “Multi-scale self-attention for text classification”, Association for Advancements in Artificial Intelligence (AAAI) (2020).

[41] Hammami, N. and M. Bedda, “Improved tree model for arabic speech recognition”, in “2010 3rd International Conference on Computer Science and Information Technology”, (IEEE, 2010).

[42] Han, S., H. Mao and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”, (2016).

[43] Han, S., J. Pool, J. Tran and W. Dally, “Learning both weights and connections for efficient neural network”, in “Advances in Meural Information Processing Systems (NIPS)”, (2015).

[44] He, K. and J. Sun, “Convolutional neural networks at constrained time cost”, in “Computer Vision and Pattern Recognition (CVPR)”, (2015).

[45] He, K., X. Zhang, S. Ren and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, in “Proceedings of the IEEE International Conference on Computer Vision”, pp. 1026–1034 (2015).

[46] He, K., X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 770–778 (2016).

[47] Hinton, G., L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”, IEEE Signal Processing Magazine **29**, 6, 82–97 (2012).

[48] Hochreiter, S. and J. Schmidhuber, “Long short-term memory”, Neural Computation (1997).

[49] Houben et. al., S., “Detection of traffic signs in real-world images: The german traffic sign detection benchmark”, in “International Joint Conference on Neural Network (IJCNN)”, (2013).

[50] Hu, J., L. Shen and G. Sun, “Squeeze-and-excitation networks”, (2017).

[51] Hu, J., L. Shen and G. Sun, “Squeeze-and-excitation networks”, in “Computer Vision and Pattern Recognition (CVPR)”, (2018).

[52] Huang, G., Z. Liu, L. Van Der Maaten and K. Q. Weinberger, “Densely connected convolutional networks”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 4700–4708 (2017).

[53] Huang, S., X. Li, Z. Cheng, A. Hauptmann *et al.*, “Gnas: A greedy neural architecture search method for multi-attribute learning”, International Conference on Multimedia Retrieval (ICMR) (2018).

[54] Huang, Y., Q. Wu, C. Song and L. Wang, “Learning semantic concepts and order for image and sentence matching”, in “Computer Vision and Pattern Recognition (CVPR)”, (2018).

[55] Ioffe, S. and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, in “International Conference on Machine Learning (ICML)”, (2015).

[56] Kadous, M. W. *et al.*, *Temporal classification: Extending the classification paradigm to multivariate time series* (UNSW, Kensington, 2002).

[57] Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, “Large-scale video classification with convolutional neural networks”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 1725–1732 (2014).

[58] Katz, L., “A new status index derived from sociometric analysis”, Psychometrika **18**, 1, 39–43 (1953).

[59] Keahey, K. and et al, “Lessons learned from the chameleon testbed”, in “USENIX”, (2020).

[60] Kopczyk, D., URL `https://dkopczyk.quantee.co.uk/hyperparameter-optimization/` (2018).

[61] Krizhevsky, A., “Learning multiple layers of features from tiny images”, Tech. rep., Citeseer (2009).

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

[63] Larochelle, H., D. Erhan, A. Courville, J. Bergstra and Y. Bengio, “An empirical evaluation of deep architectures on problems with many factors of variation”, in “International Conference on Machine Learning (ICML)”, (2007).

[64] Lawrence, S., C. L. Giles, A. C. Tsoi and A. D. Back, “Face recognition: A convolutional neural-network approach”, IEEE Transactions on Neural Networks **8**, 1, 98–113 (1997).

[65] LeCun, Y., B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard and L. D. Jackel, “Handwritten digit recognition with a back-propagation network”, in “Advances on Neural Information Processing Systems (NIPS)”, pp. 396–404 (1990).

[66] LeCun, Y., L. Bottou, Y. Bengio, P. Haffner *et al.*, “Gradient-based learning applied to document recognition”, Proceedings of the IEEE (1998).

[67] LeCun, Y., P. Haffner, L. Bottou and Y. Bengio, “Object recognition with gradient-based learning”, in “Shape, Contour and Grouping in Computer Vision”, (1999).

[68] LeCun, Y., F. J. Huang and L. Bottou, “Learning methods for generic object recognition with invariance to pose and lighting”, in “Computer Vision and Pattern Recognition (CVPR)”, (2004).

[69] Li, H., A. Kadav, I. Durdanovic, H. Samet and H. P. Graf, “Pruning filters for efficient convnets”, International Conference on Learning Representations (ICLR) (2016).

[70] Liang, M. and X. Hu, “Recurrent convolutional neural network for object recognition”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 3367–3375 (2015).

[71] Lin, Z., M. Feng and C. N. d. Santos et al, “A structured self-attentive sentence embedding”, ICLR (2017).

[72] Liu, S., Y. Garg, K. S. Candan, M. L. Sapino and G. Chowell-Puente, “Notes2: Networks-of-traces for epidemic spread simulations”, in “Workshops On Sustainability at Association for Advancements in Artificial Intelligence (AAAI)”, (2015).

[73] Liu, Y., C. Gong, L. Yang and Y. Chen, “Dstp-rnn: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction”, Expert Systems with Applications (2020).

[74] Liu, Z., W. Xu, J. Feng, S. Palaiahnakote, T. Lu *et al.*, “Context-aware attention lstm network for flood prediction”, in “International Conference on Pattern Recognition (ICPR)”, (2018).

[75] Liu et al., S., “Robust multi-variate temporal features of multi-variate time series”, Transaction on Multimedia Computing, Communications, and Applications (TOMM) (2018).

[76] Lowe, D. G., “Distinctive image features from scale-invariant keypoints”, International Journal of Computer Vision (IJCV) **60**, 2, 91–110 (2004).

[77] Lu, J. and D. Batra, “Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks”, in “Advances on Neural Information Processing Systems (NIPS)”, (2019).

[78] Lu, X., “Learning to generate questions with adaptive copying neural networks”, in “Special Interest Group on Management of Data (SIGMOD)”, (2019).

[79] Luong, M.-T., H. Pham and C. D. Manning, “Effective approaches to attention-based neural machine translation”, in “Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing”, (2015).

[80] Maas, A. L., A. Y. Hannun and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models”, in “ICML”, (2013).

[81] Martinez et al, ., “On-line learning of indoor temperature forecasting models towards energy efficiency”, Energy and Buildings (2014).

[82] Matthieu Courbariaux, M., Y. Bengio and J. David, “Low precision arithmetic for deep learning”, in “International Conference on Learning Representations (ICLR)”, (2015).

[83] Mnih, V., N. Heess, A. Graves *et al.*, “Recurrent models of visual attention”, in “Advances on Neural Information Processing Systems (NIPS)”, (2014).

[84] Mohla, S., S. Pande, B. Banerjee and S. Chaudhuri, “Fusatnet: Dual attention based spectrospatial multimodal fusion network for hyperspectral and lidar classification”, in “Workshop at Computer Vision and Pattern Recognition (CVPR)”, (2020).

[85] Moritz, E., R. Lienhart, M. Lee and L. Kennedy, “Detecting speech impairments from temporal visual facial features of aphasia patients”, in “Multimedia Information Processing and Retrieval (MIPR)”, (2019).

[86] Nair, V. and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines”, in “International Conference on Machine Learning (ICML)”, (2010).

[87] Nene, S. A., S. K. Nayar, H. Murase *et al.*, “Columbia object image library (COIL-100)”, (1996).

[88] Nene, S. A., S. K. Nayar, H. Murase *et al.*, “Columbia object image library (COIL-20)”, (1996).

[89] Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning”, Advances on Neural Information Processing Systems (NIPS) (2011).

[90] Ovtcharov, K., O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss and E. S. Chung, “Accelerating deep convolutional neural networks using specialized hardware”, Microsoft Research Whitepaper **2**, 11, 1–4 (2015).

[91] Park, J., S. Woo, J.-Y. Lee and I. S. Kweon, “Bam: Bottleneck attention module”, British Machine Vision Conference (BMVC) (2018).

[92] Pearson, K., “On lines and planes of closest fit to systems of points in space”, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2**, 11, 559–572 (1901).

[93] Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, “Scikit-learn: Machine learning in Python”, Journal of Machine Learning Research **12**, 2825–2830 (2011).

[94] Qin et al, Y., “A dual-stage attention-based recurrent neural network for time series prediction”, International Joint Conference on Artificial Intelligence (IJCAI) (2018).

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

[96] Ravindranath, M., K. S. Candan and M. L. Sapino, “M2nn: Rare event inference through multi-variate multi-scale attention”, in “SMDS”, (2020).

[97] Roffo, G., S. Melzi and M. Cristani, “Infinite feature selection”, in “International Conference on Computer Vision (ICCV)”, pp. 4202–4210 (IEEE, 2015).

[98] Rosasco, L., E. D. Vito, A. Caponnetto, M. Piana and A. Verri, “Are loss functions all the same?”, Neural Computation **16**, 5, 1063–1076 (2004).

[99] Rumelhart, D. E., G. E. Hinton and R. J. Williams, “Learning representations by back-propagating errors”, Nature (1986).

[100] Sellam, T., K. Lin, I. Huang, M. Yang, C. Vondrick and E. Wu, “Deepbase: Deep inspection of neural networks”, in “Special Interest Group on Management of Data (SIGMOD)”, (2019).

[101] Sermanet, P., S. Chintala and Y. LeCun, “Convolutional neural networks applied to house numbers digit classification”, in “International Conference on Pattern Recognition (ICPR)”, (IEEE, 2012).

[102] Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks”, in “International Conference on Learning Representation (ICLR)”, (2014).

[103] Shannon, C. E., “A mathematical theory of communication”, Bell system technical journal **27**, 3, 379–423 (1948).

[104] Silvestro Roberto, P., L. S. Maria, L. Sicong, C. Xilun, G. Yash, H. Shengyu, H. K. Jung, L. Xinsheng, N. Parth and K. Selcuk Candan, “Simdms: Data management and analysis to support decision making through large simulation ensembles”, in “20th International Conference on Extending Database Technology (EDBT’17)”, pp. 582–585 (OpenProceedings. org, 2017).

[105] Simonyan, K. and A. Zisserman, “Two-stream convolutional networks for action recognition in videos”, in “Advances on Neural Information Processing Systems (NIPS)”, (2014).

[106] Simonyan, K. and A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, International Conference on Learning Representations (ICLR) (2015).

[107] Snoek, J., H. Larochelle and R. P. Adams, “Practical bayesian optimization of machine learning algorithms”, in “NIPS”, (2012).

[108] Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting”, Journal of Machine Learning Research (ICML) **15**, 1, 1929–1958 (2014).

[109] Stallkamp, J. and et. al., “Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition”, Neural Networks (NN) (2012).

[110] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, “Going deeper with convolutions”, in “Computer Vision and Pattern Recognition (CVPR)”, (2015).

[111] Tibshirani, R., “Regression shrinkage and selection via the lasso”, Journal of the Royal Statistical Society: Series B (Methodological) (1996).

[112] Tikhonov, A. N., “On the regularization of ill-posed problems”, in “Doklady Akademii Nauk”, (1963).

[113] Tu, Z., T. Xia, C. Li, Y. Lu and J. Tang, “M3s-nir: Multi-modal multi-scale noise-insensitive ranking for rgb-t saliency detection”, in “Multimedia Information Processing and Retrieval (MIPR)”, (2019).

[114] Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention is all you need”, in “Advances on Neural Information Processing Systems (NIPS)”, (2017).

[115] Wan, L., M. Zeiler, S. Zhang, Y. Le Cun and R. Fergus, “Regularization of neural networks using dropconnect”, in “International Conference on Machine Learning (ICML)”, pp. 1058–1066 (2013).

[116] Wang et al., F., “Residual attention network for image classification”, in “CVPR”, (2017).

[117] Wiesler, S. and H. Ney, “A convergence analysis of log-linear training”, in “Advances in Neural Information Processing Systems”, pp. 657–665 (2011).

[118] Witkin, A. P., “Scale-space filtering”, in “International Joint Conference on Artificial Intelligence (IJCAI)”, edited by A. Bundy, pp. 1019–1022 (1983).

[119] Woo, S., J. Park, J.-Y. Lee and I. So Kweon, “Cbam: Convolutional block attention module”, in “European Conference on Computer Vision (ECCV)”, pp. 3–19 (2018).

[120] Xiao, H., K. Rasul and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms”, (2017).

[121] Yang, J., M. N. Nguyen, P. P. San, X. L. Li and S. Krishnaswamy, “Deep convolutional neural networks on multichannel time series for human activity recognition”, in “International Joint Conference on Artificial Intelligence (IJCAI)”, (2015).

[122] Yeh, C.-H., Y.-C. Fan and W.-C. Peng, “Interpretable multi-task learning for product quality prediction with attention mechanism”, in “International Conference on Data Engineering (ICDE)”, (IEEE, 2019).

[123] Yu, R., A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin and L. S. Davis, “Nisp: Pruning networks using neuron importance score propagation”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 9194–9203 (2018).

[124] Zaytar et al., M. A., “Sequence to sequence weather forecasting with long short-term memory recurrent neural networks”, International Joint Conference on Artificial Intelligence (IJCAI) (2016).

[125] Zeiler, M. D., D. Krishnan, G. W. Taylor and R. Fergus, “Deconvolutional networks”, in “CVPR”, (IEEE, 2010).

[126] Zhang, J., L. Hui, J. Lu and Y. Zhu, “Attention-based neural network for traffic sign detection”, in “International Conference on Pattern Recognition (ICPR)”, (2018).

[127] Zhang, Z., P. Tang and R. Duan, “Dynamic time warping under pointwise shape context”, Information sciences **315**, 88–101 (2015).

[128] Zheng, L., Y. Yang and Q. Tian, “Sift meets cnn: A decade survey of instance retrieval”, IEEE transactions on pattern analysis and machine intelligence (2017).

[129] Zheng, L., Y. Yang and Q. Tian, “Sift meets cnn: A decade survey of instance retrieval”, IEEE transactions on pattern analysis and machine intelligence **40**, 5, 1224–1244 (2018).

[130] Zheng, Y., Q. Liu, E. Chen, Y. Ge and J. L. Zhao, “Time series classification using multi-channels deep convolutional neural networks”, in “International Conference on Web-Age Information Management (ICWAIM)”, (Springer, 2014).

[131] Zoph, B. and Q. V. Le, “Neural architecture search with reinforcement learning”, in “International Conference on Learning Representations, ICLR 2017”, (2017).

[132] Zoph, B., V. Vasudevan, J. Shlens and Q. V. Le, “Learning transferable architectures for scalable image recognition”, in “Computer Vision and Pattern Recognition (CVPR)”, pp. 8697–8710 (2018).

APPENDIX A

PERMISSION STATEMENTS

Permissions regarding reuse of material from ACM papers

https://authors.acm.org/main.html

REUSE

Authors can reuse any portion of their own work in a new work of *their own* (and no fee is expected) as long as a citation and DOI pointer to the Version of Record in the ACM Digital Library are included.

- Contributing complete papers to any edited collection of reprints for which the author is *not* the editor, requires permission and usually a republication fee.

Authors can include partial or complete papers of their own (and no fee is expected) in a dissertation as long as citations and DOI pointers to the Versions of Record in the ACM Digital Library are included. Authors can use any portion of their own work in presentations and in the classroom (and no fee is expected).

- Commercially produced course-packs that are *sold* to students require permission and possibly a fee.

Permissions regarding reuse of material from IEEE papers

https://journals.ieeeauthorcenter.ieee.org/choose-a-publishing-agreement/avoid-infringement-upon-ieee-copyright/

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you must follow the requirements listed below:

Textual Material

Using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.

In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.

If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Full-Text Article

If you are using the entire IEEE copyright owned article, the following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]

Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.

"In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink."

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

APPENDIX B

NOTATIONS

APPENDIX C

ABBREVIATIONS

CNN Convolutional Neural Network 60, 65, 85

CNNs Convolutional Neural Networks 60, 64, 84, 85

DNN Deep Neural Network 60

DNNs Deep Neural Networks 60, 62, 64, 84, 85

DOG Difference-of-Gaussian 16–18, 23

DOG difference-of-Gaussian 15, 16

GSS Gaussian Scale-Space 15, 16

InfFS Infinite Feature Selection 23, 26

PCA Principal Component Analysis . . . . . xv, 23–25

SIFT Scale Invariant Feature Transform 12, 20

SVD Singular Vector Decomposition 25

UVTF Uni-Variate Temporal Features 12, 20