PSEUDO ROTATED NETS: WIDENING CNN VIA KERNELS PSEUDO ROTATION

By

Mohsen Raafat Abdel-Atty Sayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2022

PSEUDO ROTATED NETS: WIDENING CNN VIA KERNELS PSEUDO ROTATION

By

Mohsen Raafat Abdel-Atty Sayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Prof. Dr. Mohsen Abdel-Razik Rashwan

……..

Professor
Electronics and Communication Engineering
Faculty of Engineering, Cairo University

Prof. Dr. Hossam Aly Hassan Fahmy

……..

Professor
Electronics and Communication Engineering
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2022

PSEUDO ROTATED NETS: WIDENING CNN VIA KERNELS PSEUDO ROTATION

By
Mohsen Raafat Abdel-Atty Sayed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the
Examining Committee

_____
Prof. Dr. Mohsen Abdel-Razik Rashwan, Thesis Main Advisor

_____
Prof. Dr. Hossam Aly Hassan Fahmy, Advisor

_____
Dr. Omar Ahmed Nasr, Internal Examiner

_____
Prof. Dr. Khaled Mostafa External Examiner
Professor at Faculty of Computers and Information, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2022

Engineer's Name: Mohsen Raafat Abdel-Atty Sayed
Date of Birth: 22/02/1992
Nationality: Egyptian
E-mail: moh_raafat@hotmail.com
Phone: (+02) 011-20200776
Address: 68-Abou ElMahasen ElShazely,Agouza,Giza

Registration Date: 01/10/2015
Awarding Date: 2022
Degree: Master of Science
Department: Electronics and Communications Engineering

Supervisors:

Prof. Dr. Mohsen Abdel-Razik Rashwan
Prof. Dr. Hossam Aly Hassan Fahmy

Examiners:

Prof. Dr. Mohsen Abdel-Razik Rashwan (Thesis main advisor)
Prof. Dr, Hossam Aly Hassan Fahmy (advisor)
Dr. Omar Ahmed Nasr (Internal examiner)
Prof. Dr. Khaled Mostafa (External examiner)
Professor at Faculty of Computers and Information, Cairo University

Title of Thesis:

Pseudo Rotated Nets: Widening CNN Via Kernels Pseudo Rotation

Key Words: (must be 5 words only)
Machine Learning; Convolutional Neural Network; Image classification; Pseudo Rotated Kernels; Pooling Kernels.

Summary: (not more than 150 word and the summary must be in the same page)

This work aims to enhance the CNN performance through exploring the extension of its width dimension. The proposed idea is to pseudo rotate the convolutional kernels creating multiple variants from the originally trained one where each variant is pseudo rotated with a different rotation angle. Moreover, combing the pseudo rotated kernels with the pooling ones would make the network steps more towards unifying multiple of the affine transformation properties within it. This can also be viewed as if the network had been capable to self-augment the inner feature maps. To demonstrate the effectiveness of these ideas five networks were proposed that are based on two different architectures ResNet and VGG whereas they are generalized to be tested on two different data sets CIFAR-10 and CIFAR-100.

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Mohsen Raafat Abdel-Atty Sayed Date: .. /.. /…

Signature:

Dedication

To my family especially my Mother who encouraged me to continue this journey to the end.

Acknowledgments

All praise and glory to almighty **ALLAH** for all the blesses, guidance and opportunities given through this thesis journey. Throughout this journey many people had shown help and support so I would like to express my deepest thanks and appreciation for all of them. I would like to dedicate a special appreciation and gratitude to my advisors **Prof. Dr. Mohsen Abdel-Razik Rashwan** and **Prof. Dr. Hossam A. H. Fahmy** for their patience, guidance, inspiration and encouragement. Without their help and support this work won't have been possible. Lastly, I would like to thank my beloved family for their compassion, support, love and understanding.

Table of Contents

List of Tables

List of Figures

Nomenclature

Abbreviation	Description
AI	Artificial Intelligence
AR	Augmented Reality
API	Application Programming Interface
BM	Boltzmann Machine
BN	Batch Normalization
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CTC	Connectionist Temporal Classification
DARPA	Defense Advanced Research Projects Agency
DRAM	Data Random Memory Access
DBM	Deep Boltzmann Machine
DNN	Deep Neural Network
FFT	Fast Fourier Transform
FCL	Fully Connected Layer
GRU	Gated Recurrent Unit
GMM	Gaussian Mixture Model
GPU	Graphical Processing Unit
HMM	Hidden Markov Model
IoT	Interne of Things
LCN	Local Contrast Normalization
LRN	Local Response Normalization
LSTM	Long Short Term Memory
LTD	Long Term Depression
LTP	Long Term Potentiation
MFCCs	Mel Frequency Cepstral Coefficients
ML	Machine Learning
MLP	Multi-Layer Perceptron
CuDNN	NVIDIA CUDA Deep Neural Network library
NLP	Natural Language Processing
NN	Neural Network
NoC	Network on Chip
SNN	Spiking Neural Network
STDP	Spike Timing Dependent Plasticity
TPU	Tensor Processing Unit
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
TFLOP	Trillion Floating Point Operation Per Second
VR	Virtual Reality

.

Abstract

In the Data explosion era, terminology like “Big Data” had been commonly used as the world had been connected and digitalized through the wide availability of personal computing platforms with their internet connection, rapid spread of the mobile platforms, popularity of the social media applications and the start of Internet of Things platforms paradigm accompanied by the invention of smart devices that are almost utilized in all aspects of today life from wearable devices to kitchen appliances. All of the aforementioned, had resulted in a daily generation of huge amount of digital data such as documents, videos, image and speech. These type of data are distinctly characterized by their personal flavor gaining the attraction to use the Machine learning methods to extract useful insights, predictions and information from them. Moreover, around 70% of these data are images and videos increasing the requirement to enhance the computer vision tasks. Convolutional Neural Network which is a sub domain of Machine learning had been the key player in today enhanced computer vision tasks. Thanks to its distinct features such as weight sharing, feature map, channel pooling and receptive field.

This work explores boosting the Convolutional Neural Network performance by means of width extension. This is done through two main ideas. Firstly, pseudo rotated kernels where the originally trained kernels are rotated with different pseudo rotation angles to generate multiple variants from them. Secondly to attach the pooling kernels to the convolutional layer. This allowed the network to approach several affine transformation properties. Clearly, it boosts the translation and rotation property by providing a set of arbitrary chosen pseudo rotated kernels while it promotes the scaling property through the arbitrary reduction of grid size. Moreover, all these kernels combined together provide the network with a capability to scale and rotate the feature map within each convolutional layer increasing its translation invariance property robustness whereas the network had some built-in self-augmentation methods. To demonstrate the performance improvement five networks were proposed based on two different architectures where three of them are based on ResNet while the remaining two are based on VGG. As well as, challenging their performance impact by testing them on two different data sets the CIFAR-10 and CIFAR-100.

Chapter 1 : Introduction

1.1. Data Explosion Era

At the start of Big Data era, wide availability of personal computing platforms connected to the internet had led to a digitalized world where huge amount of digital data such as documents, videos, image and speech were generated daily.

The explosion of data era had extended more with the inventing of the mobile platforms and the rising of the social media applications which rapidly had gained popularity among people all over the world leading to the generation of more digital data and information with a special personalized nature. For instance, according to [14] Facebook generates over 10 Petabyte (PB) log data per month and Taobao.com, the largest online retailer in China, generates tens of Terabyte (TB) data every day.

Moreover, the start of the next wave of connecting and digitalizing the world through 5G communication technologies and Internet of Things (IoT) platforms allowed the invention of edge computing devices which are sensor rich based devices with a high speed internet connectivity giving it the capability to exchange information with powerful computing servers (i.e. data centers). Hence, generation more and more data with a personalized flavor.

All the aforementioned had driven the need of statistical and analytical solutions to be able to solve the learning problem aroused from these vast data to extract useful insight and knowledge form them

Conventional approaches which relies on domain experts to express the problems analytically, transform the raw data into useful features and representation then hand craft the solution had failed to deal with this tremendous growth in the scale of data with its personalized nature as it requires an explicit knowledge about the given domain limiting its ability to solve more complex problems in which the features and knowledge representation can't be explicitly expressed as they are implicitly inherited with in the raw data.

Meanwhile, Artificial Intelligence(AI) solutions especially its Machine Learning (ML) sub-domain had provided a leap over these tremendous data where it allows automatic features and information extraction as well as the acquisition of useful insights, predictions and decisions from this huge amount of data without the need of formally expressing the features nor the representation resulting in approaching more complex problems such as medical diagnosis and speech transcription.

ML significant value appears in its ability to overcome the personalized nature of how modern data are generated meanwhile maintaining the privacy of these data through preprocessing to remove any personal labels.

This edge of ML over the conventional solutions regarding its ability to deal with the tremendous growth in the scale of data and information with its associated learning problem had developed a prominence demand on ML. This demand had resulted in a respond from the ML community which can be shown in Figure 1 where more than 50 ML papers appear daily on arXiv.org alone and their rate of growth is almost doubling every two years which can be compared to Moore's Law.

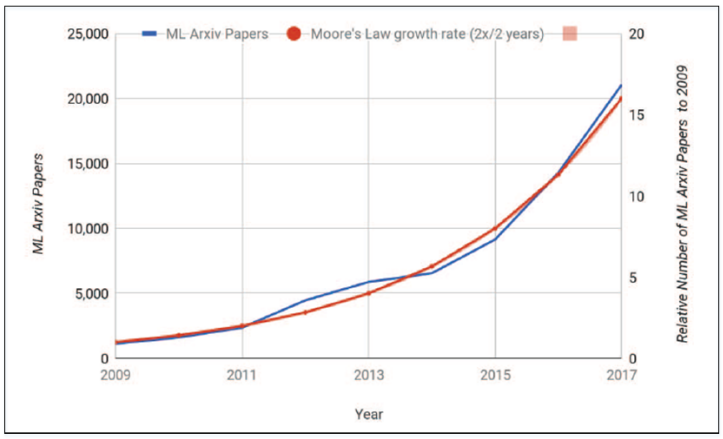

Figure 1 : The number of ML papers posted on arXiv.org per year from [13]

1.2. Artificial Intelligence and Machine Learning

Learning problem according to [1] can be described as the problem of executing a task and optimizing its performance metric through training experience

Meanwhile, Representation Learning according to [2] can be viewed as the set of methods in which the raw data is fed to the machine then the machine can automatically distinguish all the features and representation required for acquiring the useful knowledge needed for the next action whether it was detection, classification or any other tasks

Artificial Intelligence (AI) according to [3] is any agent device that can become conscious about its surrounding environment and can take the actions that maximizes its ability to achieve its goals. The popularity of AI among the scientist and engineers is increasing due the achievements and the breakthrough performance driven from its Machine Learning (ML) sub-domain. A Venn diagram to illustrate the relation between AI and its ML sub-domains can be shown in Figure 2.

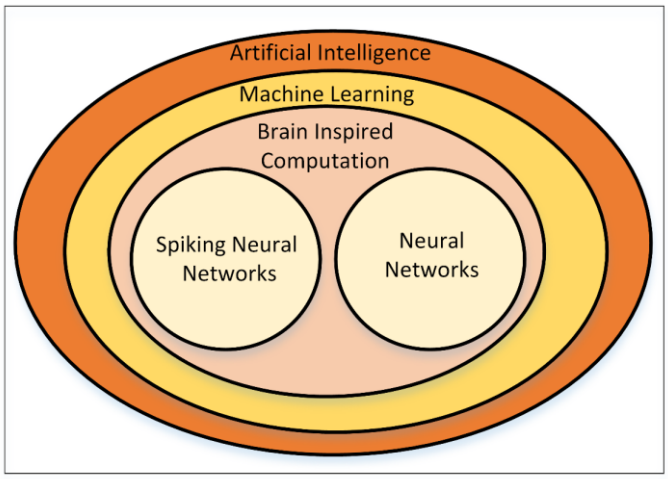

Figure 2 : Venn diagram between AI and its ML sub-domains

ML was first quoted by Arthur Samuel in 1959 as giving the ability to the machine to learn without being explicitly programmed to do that consequently allowing the

creation of programs to do some activities through leaning and training experience, on contrast to hand crafted programs which their activities and behaviors are defined in a hard coded style. ML enables the emulation of how humans learn, adapt and make decisions. This lead to the designing of programs that has the ability to learn the required actions based on the knowledge learnt form from raw data directly. ML can be seen as programing by example where previous experience shall contribute to the gained knowledge affecting the future actions.

ML ability to solve a problem with a high performance generally depends on two factors: the data set availability compared to the problem and the computational infrastructure available

The complexity of the problem with its inherited required features to be learnt affects the amount of data required and the rule of thumb here is that as the amount of data increases, the ability to capture more patterns and features automatically increase and hence the quality of results increase proportionally.

On the other hand, enormous data set shall require a giant network that shall essentially come with a huge computational cost penalty that may limit the ability to train it if the required hardware infrastructure isn't available. Nowadays, training a modern network may require two high end GPUs that are capable to perform multiple TFLOPS operations.

1.3. Machine Learning Algorithms

Generally, ML algorithms according to [1,2,16] can be divided as shown in Figure 3 into four categories: Supervised Learning, Unsupervised Learning, Semi Supervised Learning and Reinforcement Learning

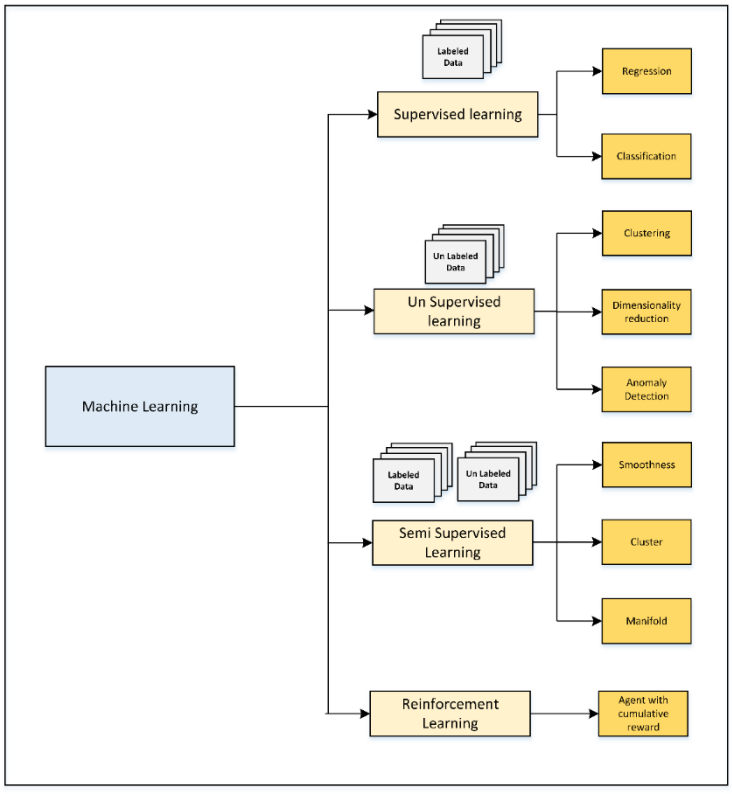

Figure 3 : ML different learning styles

1.3.1. Supervised Learning

Supervised learning methods requires the availability of a labeled data set in which each input is tagged with its desired output. The objective is then to feed the machine with the input data and train its prediction to match the reference.

The learning is described as supervised since there is a known reference output that acts as a supervisory guidance for the whole training assisting in the reduction of the gap between the predicted output from the network during training and the actual one.

This type of learning can be divided into two main categories known as Regression and Classification. The first tries to identify the most likely function that can fit all the data within the data set, while the latter attempts to find the best fit class for the data from a set of given classes.

1.3.2. Unsupervised Learning

Unsupervised learning doesn't require a labeled data set instead it is fed with data without explicit labelling or desired output. Thus, there is no right or wrong outputs instead it is subjective to the application itself.

The objective is to find common statistical and structural properties of data through automatic extraction of the underlying features and patterns enabling their cluster into groups based on the correlated features extracted during training.

There are three main categories in this type of learning which are clustering, dimensionality reduction and anomaly detection.

1.3.3. Semi Supervised Learning

Semi supervised learning includes a mixture from supervised and unsupervised learning where both labeled and unlabeled data are used during the training. Usually, used when the amount of labeled data is small and hence, extracting patterns and features from them isn't satisfying, meanwhile labeling the unlabeled data requires an extensive time and the availability of domain experts.

The objective is to augment the unlabeled data with the labeled one through the creation of the data cluster using the unlabeled data and using the labeled data to identify the clusters. There are three main categories in this type of learning based on the assumption used during the training which are smoothness, cluster and manifold.

1.3.4. Reinforcement Learning

Reinforcement learning is different from all the aforementioned in which it is defined in the terms of having an agent that tries on its own to interact with the surrounding environment based on trial and error approach with a cumulative reward that guide the agent to learn the right decision on its own instead of being explicitly trained. The agent shall have two states the start and the end. Between the two states there is different routes and actions that may cause success or failure to execute the task and reach the end state. Hence, the agent receives a reward when moving towards the end in the optimal route while it doesn't receive anything upon failure.

The objective is then to achieve the target and move from the start to the end state with maximum cumulative reward. It is an iterative method that depends on the past feedback and the ability to span new approaches to reach the goal.

1.4. Machine Learning Brain Inspired Computation

Brain inspired computation is a sub-domain of ML as shown in Figure 2 that is trying to mimic some basic operations of the brain according to the understanding of how the brain operates nowadays, with the objective to emulate the brain in some processing aspects rather than creating a human brain.

The current biological structure and characterization of the brain can be shown in Figure 4.

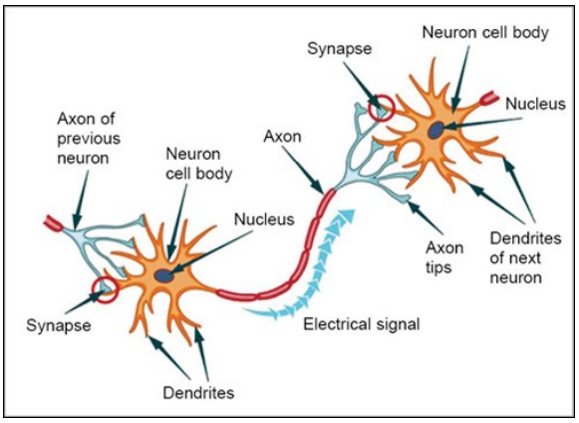

Figure 4 : Brain biological structure from [5]

The brain is composed of a neuron which is considered to be the main computational element. This neuron is connected with other neurons through dendrites and axons. Both dendrite and axon can be referred to as the activation of the neuron as dendrites allow input signals to enter the neuron meanwhile the axons allow the signals to exit out of it. When a dendrite and axon are connected together they form a synapse. A key feature of the synapse it allows scaling the signal associated with it. This scaling can be viewed as a weight value and the brain is believed to be able to learn through the ability to change these weights in respond to different input stimulus.

The way of brain learning process is the key inspiration of the ML Brain inspired computation where it is based on the continues adjustment of the weights in response to the training stimulus while its infrastructure referred to the number of neurons and the connection among them remains fixed which maps to the network structure.

The Brain inspired computations can be divided into Neural Network (NN) and Spiking Network

1.4.1. Neural Networks

Neural Networks(NN) are inspired from neuroscience where it tries to make analogy with the biological structure of the brain where the computations take part within the neuron of the network. These computations can be viewed as a neuron firing to generate its output by applying a nonlinear function on a weighted sum of the inputs with an optional addition of a bias. With the synapses being modeled through the adjustable weight associated with each input signal allowing its scaling during the training experience.

The computational flow of the NN is usually visualized using a directed acyclic graph (DAG) [4,12,21,23] as shown in Figure 5.

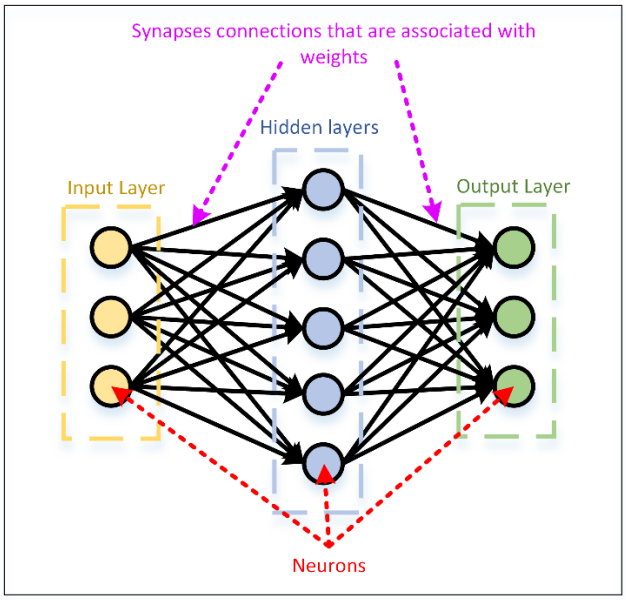

Figure 5 : Simple NN structure with one Hidden Layer

The vertex illustrates the neuron, the directed edge demonstrates the synaptic connection between the neurons and hierarchical structure of the neurons describes the organization of the network.

This multi-layer hierarchy allows the first few layers to act as low feature extractor (i.e. extracting the edges) while enabling the last few layers to represent the high level feature (i.e. representing the complex contour) and in between allows the processing of the extracted features to their high level representation.

The computational flow starts with the neurons of the first layer noted as input layer accept the input values, applying the nonlinear function and propagating the outputs to the middle layers. The middle layers are noted as hidden layers and based on the network structure whether the hidden layer has a depth of a few layers it can be noted as a Shallow Neural Network or its depth has many layers it can be noted as Deep Neural Networks (DNN). Consequently, the hidden layer neurons accept the inputs from the input layer and perform the same operation from applying the nonlinear function and propagating its outputs to the output layer which shall be the final output of the network.

NN shall comprises two phases along its usage life time: Training phase and Inference phase.

The training phase is the learning phase in which network development takes place from defining the type of network, number of layers and continuously manipulating the weights to meet the required performance on a given application.

On the other hand, inference phase is the prediction phase in which the network is deployed in production and used in a feed forward manner.

The high popularity of NN nowadays can be argued to the superior performance of its DNN family of networks especially the Multi-Layer Perceptron (MLP), Convolutional Neural Network(CNN) and Recurrent Neural Networks(RNN) where these networks were able to suppress the human level performance on various tasks such as ImageNet recognition [29] and Atari 2600 video games [113]. Furthermore, these types of networks represent 95% of NN inference workload in google datacenters according to [12].

1.4.2. Spiking Neural Networks

Spiking Neural Networks (SNN) try to pursuit a biological brain inspired paradigm in a different fashion from the traditional ML neural networks, where the first is directly inspired from neuroscience in the way it encodes, transfer and processes the data while the latter mimic the relationship between the neurons in a more remote way using the activation of a weighted sum of input data through a nonlinear function.

SNN which can be shown in Figure 6 is based on asynchronous communication between different neurons allowing time dependent information transfer through train of pulses where the information is coded in the form of spikes. Meaning that the neuron shall have the capability to extract information from an encoded timing pulse specifically the pulse width, amplitude and the time of arrival of the pulse relative to other pulses. Consequently, when a neuron spikes it inhibits all other neurons, emulating the presence of inhibitory connections and the spiked neuron enters a refractory phase where it ignores any coming spike. This spiking nature is more readily to receive and operate on real world data since they are usually pulse oriented with a time varying nature. In addition, they are most suited in low power applications as the spiking rate may be as low as few tens of Hertz. However, SNN is still not competitive with the accuracy results achieved by state of art of the ML neural networks on different datasets

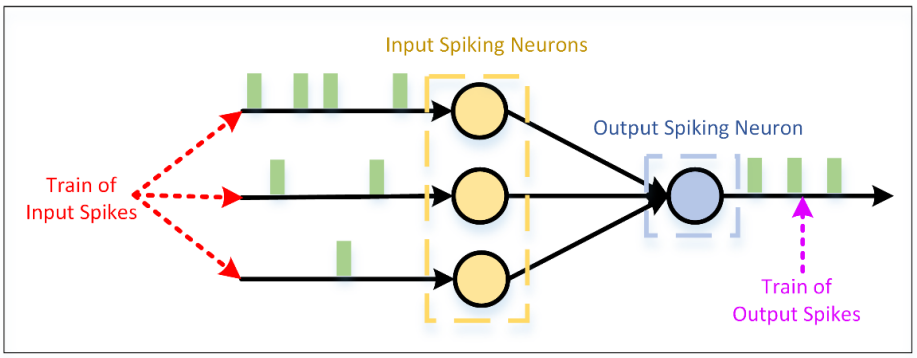

Figure 6 : Simple SNN

SNN training is challenging since their discontinuous spiking nature is not suitable for the backpropagation algorithm which requires the model to be differential to generate the errors in form of the gradients. One approach to train these networks is to use the Spike Timing Dependent Plasticity (STDP) learning method. An unsupervised learning which relies on the spiking timing whether pre or post the synapse to obtain the causality between input and output spikes. This causality is obtained through detecting when a neuron fires after the arrival of the input spikes. If it fires soon this would likely map that synapse had an impact and thus needs to be boosted, meanwhile if it fires later after the arrival of the input spike this would likely map that the synapse had no role in this firing and thus needs to decreased. The first process is referred to as Long Term Potentiation (LTP) while the latter is Long-Term Depression (LTD). STDP is accompanied with the need of Homeostasis process. Homeostasis is a process used to balance the distribution of the information among different neurons through the firing threshold adjustment. Meaning that, if a neuron fires frequently they are punished by increasing their firing threshold. On the other hand, if they fire infrequently they are inculcated by decreasing

their firing threshold. This shall ensure that all the neurons shall contribute in the output generation enhancing the network performance by squeezing out all its capabilities.

Prominent examples of SNN approach are IBM through its TrueNorth chip [57] which has one million programming neurons and 256 million configurable synapses, Qualcomm through its Zeroth processor [56] and Manchester University's SpiNNaker[55].

1.5. Machine Learning Stack

ML design space is similar to any other Hardware Software Co-design space can be viewed in the form of stacked layers one on the top of the other where each layer is considered with a portion of this space allowing the focus on its constraints, required specifications and available optimization techniques.

This Stack methodology is simply the divide and conquer approach which is used to divide a big problem into a series of smaller ones that can be easily understood, constrained and optimized such that when collecting all the parts together the overall performance is maximized

As shown Figure 7 in this stack layer can be divided into five layers: Application Layer, Architecture Layer, Software Layer, Hardware Layer and Benchmarking and comparison Layer.

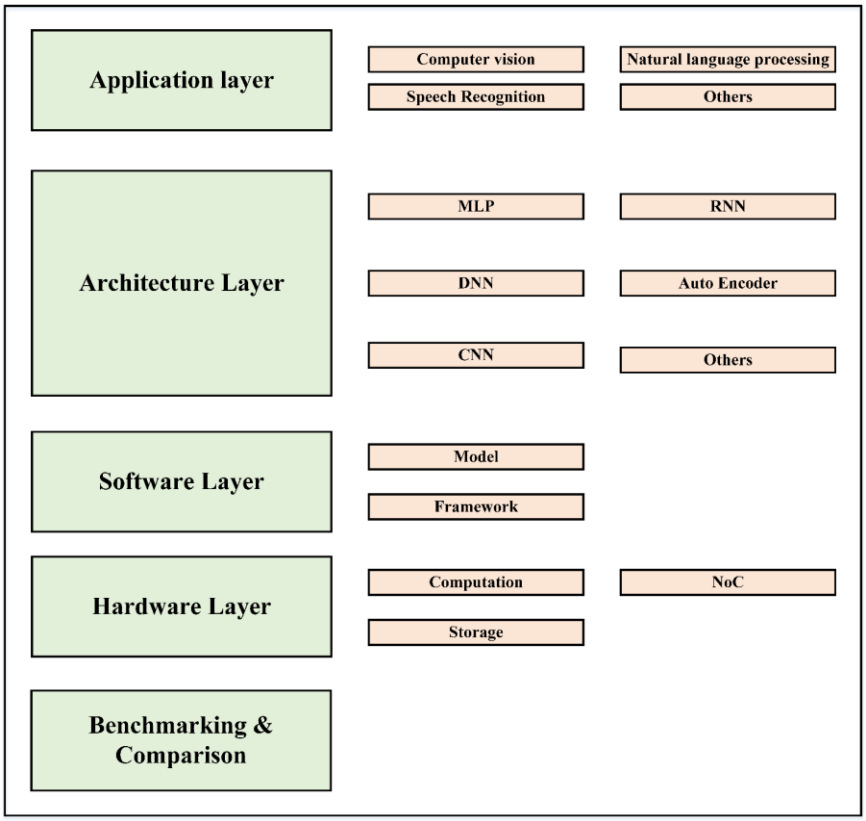

Figure 7 : ML Design Stack Overview

1.5.1. Application Layer

The Application Layer shall define the required problem to be addressed by means of ML methods.

Well Known Applications include Computer Vision, Speech Recognition, Natural Language Processing (NLP), Recommendation Systems, Robot Control, Cosmology, Social science and many others.

1.5.2. Architecture Layer

The Architecture Layer shall define how the data is organized, prepared to be processed, required computational flow to generate the output as well as the memory requirement to process all the data.

Widely used Architectures includes Multi-Layer Perceptron (MLP), Deep Neural Network(DNN), Convolutional Neural Network(CNN), Recurrent Neural Network(RNN), Auto Encoders, Boltzmann Machine(BM), Deep Boltzmann Machine (DBM), Linear Regression, Logistic Regression, K-nearest Neighbors, Support Vector Machine (SVM) and many other architectures.

1.5.3. Software Layer

The Software Layer shall define the network modeling techniques in which the data flow defined in the Architecture Layer is translated into basic operations facilitating the characterization of the network performance, obtaining useful insights to identify potential are of improvement and optimizing the network through evolving techniques such as network precision reduction, activation statistics monitoring, network pruning and others.

Also, it shall define how the model shall be implemented either using low level languages or using framework. Frameworks are currently attracting attention as it abstracts the implementation of software model using high Level libraries instead of starting from scratch using basic operations, thus reducing the time of implementation significantly and leveraging the accumulated optimization knowledge in the field of ML across the whole community.

1.5.4. Hardware Layer

The Hardware Layer shall define the hardware architecture to implement the software model defined in the Software Layer including how the processing of data shall be done, the architecture of the data processing units and whether they are going to use exact computation or Stochastic/Approximate computations as part of operation accuracy versus power trade off.

Besides, the Hardware Layer shall define the memory hierarchy, associated policy and any applicable Near Data Processing. Furthermore, given how the current networks are computational hungry requiring many processing units to operate together, it shall define the Network on Chip (NoC) architecture including the infrastructure that allows different processing units to exchange data and allow data transfer across different memory hierarchy whether they were On-chip or Off-chip.

1.5.5. Benchmarking and Comparison Layer

Benchmarking and Comparison Layer shall define the community performance metric and what are the key aspects when comparing various designs and techniques relative to each other to achieve a fair method highlighting the different trade-offs.

1.6. Organization of the thesis

This work shall focus on Image Classification enhancement through Supervised Learning with a focus on the Architecture Layer mainly the CNN architectures.

The remainder of this thesis is organized as follows:

- Chapter 2 : Provides the literature survey encompassing the different layers within the ML design Stack
- Chapter 3 : Shows a detailed survey considering the popular CNN networks and their progress with respect to the ImageNet competition.
- Chapter 4 : Explores different ideas to extend the width of the convolutional layer and mainly introduces the Pseudo Rotated Kernels
- Chapter 5 : Generalize the Pseudo Rotate kernels through proposing five networks based on two different architectures as well as testing them against two different data sets
- Chapter 6 : Compares the proposed networks with the literature ones as well as benchmarking them relative to top performing networks proposed for each data set.
- Chapter 7 : Summarize the thesis work and discuss the future work

Chapter 2 : Machine Learning Stack Literature Review

This chapter presents the literature review for the application, architecture and software ML stack layers meanwhile the hardware and the benchmarking and comparison layers are considered as out of scope. Clearly, it surveys each one of the in scope layers within the stack to show its progress over time as well as exploring the various approaches applied to enhance the ML different performance metrics within each layer in addition to demonstrating the different available tradeoffs.

2.1. Application Layer

DNN a sub-domain of ML had shown a remarkable performance across a wide range of fields, outperforming the previous state of art techniques and accomplishing a breakthrough results. For example, starting from the AlexNet [65] at 2012 where the error at ImageNet competition [64] was around 25%, it had driven the error down to 3.5% through the ResNet architecture [29] suppressing the human level accuracy in the image classification tasks. Moreover, according to [28] using DNN in speech recognition had led to the reduction of word error rate by 30% when compared to other conventional methods which is the biggest gain in the speech field in the last 20 years. Needless to mention, mastering the Go game and defeating a human champion [30].

In this layer, the focus shall be on the computer vision and speech recognition applications discussing how the DNN is leveraged among these applications as well as their popular associated public data sets. Admittedly, public data sets were a crucial key for the development and training of new network architectures as well as enabling fair comparison between them

2.1.1. Computer Vision

2.1.1.1. General Overview

In the era of data explosion, video is considered to be the dominant type of data generated nowadays, in fact according to [58] it contributes with over 70% of today's internet traffic. Moreover, according to [59] more than 800 million video hours for video surveillance is collected daily worldwide.

Hence, there is an urgent need for computer vision tasks such as image classification and segmentation, object detection, localization and tracking and action recognition to analyze as well as extract useful information and insights from this huge amount of data. Moreover, the enhancement of these tasks is considered a key feature for enabling a set of new applications such as augmented reality (AR), virtual reality (VR) and robotics.

2.1.1.2. Image Classification

Image Classification is the most common and primary task within the computer vision; Furthermore, it forms the basis for another tasks such as object detection and localization. It involves identifying the most likely class a given image shall belong to from an entire set of classes.

DNN through its CNN variant had progressed starting from LeNEt-5[60] which was designed for simple grayscale digit recognition until the ResNet[29] that were applied on ImageNet competition[64] with around 1.2 million color resolution achieving an accuracy 3.5% suppressing the human level performance. Undoubtedly, their distinct performance had enabled their usage in more critical applications such as medical analysis one where they are used to detect whether a disease exist or not. For instance, they are used in diagnosis of different kinds of cancers from brain [78] to skin [79] and breast [80] with an achieved competitive performance to the human proficient.

Popular data sets for image classification are MNIST [62], CIFAR [63] and ImageNet [64].

MNIST as shown in Figure 8 is a handwritten digit data set introduced in 1998, composed of ten classes (equivalent to ten digits) with 60,000 training image and 10,000 test image with a total size of 50MB and each image is grayscale 28 x 28 pixel. Actually, it is considered to be a handy data set.

Figure 8 : MNIST data set examples

CIFAR is a subset of the 80 million Tiny Image data set introduced in 2009. It has two variants as shown in Figure, the first is CIFAR-10 shown in Figure 9(a) which is composed of ten classes of various objects and the second is CIFAR-100 shown in Figure 9(b) which is composed of hundred mutually exclusive classes with more objects included. Both variants have 50,000 training image and 1000 test image, where each image is a colored 32 x 32 pixel. The total size of the data set is 170 MB.

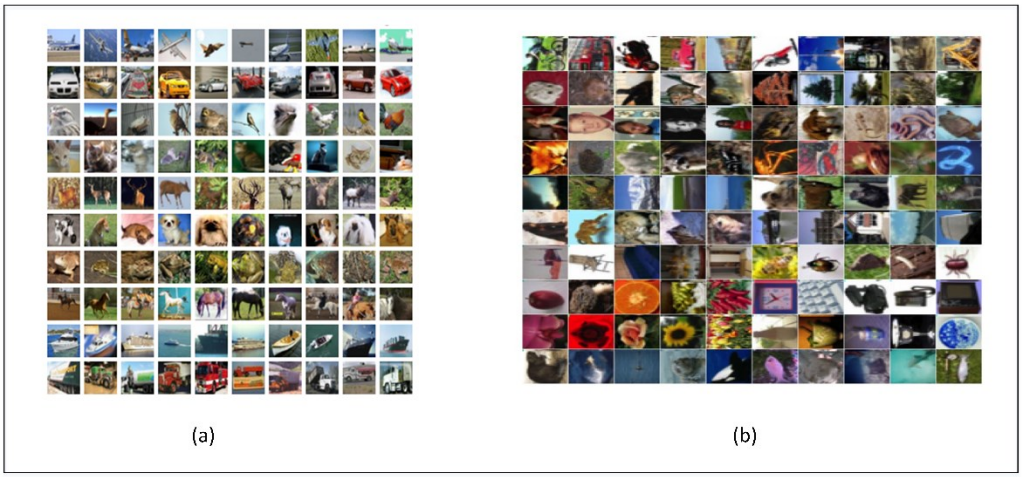

Figure 9 : (a) CIFAR-10 data set examples (b) CIFAR-100 data set examples

ImageNet as shown in Figure 10 was introduced in 2010 but stabilized in 2012, composed of 1000 classes with 1.3 million training image, 100,000 test image and 50,000 validation image. Each image is colored 256 x 256 pixel. Moreover, the ImageNet was first to introduce the Top-5 and Top-1 error metric. Top-5 error is calculated by considering that the classification is correct if any of the top five scoring categories are the correct category, meanwhile the Top-1 error considers only the top scoring category as the correct one.

Figure 10 : ImageNet data set examples

Recently, Google shared its Open Images data set [68] data set composing of 6,000 classes with over 9 million images, spanning 6000 categories

2.1.1.3. Object detection

Object detection is a multi-task application; composed of classification and localization tasks. The first one focuses on the identification of the instances of an object that belong to a specific class within the image, meanwhile the latter focuses on estimating the location of these instances.

With the emergence of DNN and the invention of Regions within CNN framework (R-CNN) [82], a significant improvement had been achieved allowing real time accurate object detection applications. Its basic idea is to create a unified framework that divides the image into a set of candidate windows, classifies them by means of a CNN and finally labelling them into rectangular bounding boxes to generate the final results directly without any post processing. The region based CNN unified framework paradigm had continued to improve through set of proposals including Fast R-CNN [79] which jointly optimizes classification and bounding box regression tasks, Faster R-CNN [81] which adds a subnetwork to generate candidate regions and YOLO [80] a fixed-grid regression approach.

Face recognition is one example for the object detection tasks where its objective is to identify and locate the face regions with the ability to cope with faces unique structures and characteristic such as face parts distributions and skin color. Moreover, it shall be able to handle the visual variations including pose changes, illumination changes and occlusions. DNN through its CNN sub domain had brought a change in this application through the proposed state of the art networks such as Google's FaceNet [83] which is based on training the CNN with a triplet loss function to allow the network to learn to cluster the face representation of the same person, Facebook's DeepFace [84] where it models the face in a three dimensional shape then align it to a frontal pose then feed to a CNN composed of a single convolutional layer, a single pooling layer, three locally connected layers and two fully connected layers and OpenFace [85] an open-source face recognition tool.

Popular data sets for object detection are PASCAL VOC [66] and Microsoft COCO [67].

PASCAL VOC is introduced in 2005 and stabilized in 2012, composed of 20 classes with 11,000 images, 27,000 object instances and 7,000 of them had detailed segmentation.

Microsoft COCO is composed of 91 classes with 2.5 million labeled instances in 328,000 images. Compared to ImageNet, it has fewer classes, however it has more images per class and more labels which is rigorous for contextual information extraction and localization.

2.1.1.4. Action and Activity Recognition

Action and activity recognition is one of the most challenging tasks that has a wide range of applications including robotics, human computer interaction and video surveillance. It involves identifying human activities from an image or video sequences which can be classified into gestures, human to object interaction, human to human interactions, events, group actions and atomic actions.

The hardness of this task is due to the requirement to solve the distorted and translated features among different patterns that belong to the same action category which arise from several problems such as occlusion, changes in scale, viewpoint, illumination, and background clutter.

After DNN had made a breakthrough in image classification, it started to impact the activity recognition achieving new state of the art results but still away from the level of impact brought to image classification. The current state of art is a dual architecture that combines both CNN and LSTM [87,88,89]

YouTube data set [69] is one of the data set allowed to be public recently from Google. It spans 4,800 classes with 8 million videos (0.5 million hours of video)

2.1.2. Speech Recognition

2.1.2.1. General Overview

Modern computing platforms are now featured with a voice assistant user interface such as Google Now, Apple Siri or Microsoft Cortana. These type of interfaces are based on automatic speech recognition systems which necessary are required to provide a continuous real time speech recognition that shall be speaker independent as well as capable to cover a large vocabulary. Thus, improving the performance of speech recognizers is critical for the overall user experience. Moreover, with current trend of Internet of Things (IoT) platforms where the invention of smart devices is exploding in almost all aspects of today world from wearable devices to kitchen appliances over to children's toys had increase the need for a neat human computer interface. Traditional interfaces like keyboard or mouse are not suited with these kind of devices due to the physical structure shrunk of these devices making typing a tough task, meanwhile an elevating approach is the speech interface where the voice is used as the interaction method to give commands and exchange information; increasing more the need for high performance speech recognizers.

Speech recognition is a sub task within the speech processing applications where it is required to identify word and phrases sequences uttered in a continuous fashion and transform them into a machine understandable format. The speaker voice is captured by a microphone in the form of acoustic signal then converted to a set of words where it can be the final result if the application is speech recognition or it can be used to feed further linguistic processing such as speech synthesis.

2.1.2.2. Historical Background

Gaussian Mixture Models (GMMs) that are based on hidden Markov models (HMMs) had been dominating the speech recognition for a long time with a few attempts to apply the traditional neural networks, however its achieved performance has been lagging behind the state of art of GMM-HMM methods at that time. The GMM-HMM methods are based on approximating the speech signal into a piecewise short time stationary signal where it can be considered as a stationary process, hence enable the usage of Markov model for many stochastic processes. Meanwhile, Each HMM uses a Gaussian model for representing the spectral of sound wave. This combined method enables the extraction of the temporal patterns of the speech.

DNN started to have an observable impact in speech recognition in 2013[7], after the major research groups worldwide including IBM, Microsoft, Google and Baidu had shown that applying DNN on large speech recognition tasks using the raw speech spectral features of the spectrogram away from Mel-frequency cepstral coefficients (MFCCs) features had shown great success. From then on, DNN had started to become the main stream method for both the academia and the industrial speech recognizers. The quick adaption of the DNN based speech recognizer across the entire speech recognition community can be regarded to the minimum change required in the speech decoder through the usage of senones as the output from the DNN, dramatic performance enhancement compared to GMM-HMM systems and the availability of large amount of data required to train this networks.

2.1.2.3. DNN State of the art architectures

Traditionally state of the art DNN architectures were trained by dividing the speech recognition systems into three separate components the acoustic, pronunciation, and language models where each component is trained separately with a different objective. The acoustic model is typically trained to extract the context dependent phonemes with the assist of an alignment method, pronunciation model trained to map the sequences of phonemes produced by the acoustic model into word sequences through a linguistic model developed by domain experts and the language models are trained on huge amount of text data to estimate probabilities of word sequences.

Current state of the art architectures is focusing on end to end trained speech recognition systems. End to End shall refer to transforming all the speech recognition models to a single sequence to sequence model where the acoustic, pronunciation, and language models are trained jointly and optimized to achieve the required performance metric typically the overall system word error rate and hence the objective of the training is to map directly the sequence of raw speech waveforms to sequence of words without any need of alignment between the input waveform and the output characters. This sequence to sequence model is typically composed of encoder and decoder to overcome the problem of variable input and output sequences length. The encoder maps the sequential variable input length to a fixed length vector while the decoder utilizes this fixed length vector to generate a variable output sequence length. To attempt the end to end training goal various methods are applied, for instance the connectionist Temporal Classification (CTC), RNN transducer, attention based models and hybrid CNN-RNN architectures.

2.1.2.3.1. Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) is an end to end training method that doesn't require a frame level alignment of target labels for training utterance where it attempts to emit any label or no label at every time step thus segmenting the alignment into a distribution of possible regions between the input and output sequences meanwhile every label is emitted into a single time step fashion. It essentially needs the set of target labels to be augmented with an additional blank symbol as well as the existence of intermediate label representation to allow labels repetition and blank labels occurrence without identifying them as a target output (i.e. emit no output label)

2.1.2.3.2. RNN transducer

RNN transducer which was introduced in [92], is an extension for the basic CTC method where it combines CTC end to end method with a separate language model based on LSTM (Long Short Term Memory) an RNN architecture variant which improves the memory effect of RNN. LSTM can deal with the sequential nature of speech since the current hidden state can be function in all the previous and future hidden states, thus it can exploit the frame information dependency whether the past dependency in the case of a unidirectional LSTM or both past and future in case of bi directional. In this method, the acoustic and language models are jointly trained where the CTC is used as an acoustic model to determine the distribution over phones sequences based on the acoustic waveform, meanwhile the transducer identifies the phoneme based on the proceeded ones. This method allows the network to predict the output based on its previous output sequences and it its current location within the input sequence. The transducer is accompanied by a decoder either a bean search one or prefix search or deciding based on the active output at every time step.

2.1.2.3.3. Attention based models

Attention based models for instance as [93], is composed of an RNN encoder named listener and an attention based RNN decoder named speller. The encoder transforms the acoustic speech waveforms to higher level features while the decoder converts these features into output characters by performing a conditional prediction to emit the target characters based on the full history of previous predictions and acoustics using the attention mechanism. This method differs from the RNN transducer that it combines the prediction network and the acoustic model into a single model instead of separate models that are trained jointly. Moreover, the key improvement of this method it generates the character sequences without making any independence assumptions between the characters in contrast to the CTC which assumes conditional independence between input acoustic frames. The attention mechanism is required to feed the decoder with selectively chosen information relevant to the current emitted output allowing the creation of a skip connections that can effectively flow the data through the RNN. On one hand, this would improve the performance as well as reduce the required computations, in addition to reducing the overfitting problem by preventing the network from memorizing the transcripts and force it to pay enough attention to the relevant information.

2.1.2.3.4. Hybrid CNN-RNN architectures

The recent CNN developments driven by the vision community and its associated outstanding performance had led to experimental usage of hybrid CNN-RNN architectures mainly the CNN-LSTM flavor architectures within speech recognition architectures. Historically, CNN was combined with HMM-GMM in a hybrid model where the HMM-GMM force a frame level alignment before the CNN can be trained to generate the required targets. In other words, the HMM-GMM perform the temporal modeling while the state predictions were generated using the CNN. However, given the fact that LSTM had become the default practice nowadays when dealing with data with sequential nature as speech recognition, combing both strengths of CNN and RNN into a hybrid architecture would be a promising approach. On one hand, CNN can effectively exploit the spectral structure locality in the feature space. Moreover, through its frequency dimension weight sharing as well as using the pooling layers helps to tradeoff between vocal tract length invariance and the trajectory speech sound differentiation as well as reducing the spectral variations within the acoustic features. On the other hand, LSTM is well known for its temporal modelling capability. Moreover, when accompanied with CTC the end to end goal had become feasible while setting a new bar for the achieved performance. Thus, combining them is a promising approach where the a few CNN layers are used to reduce the spectral variation of the input then feed the extracted features to a deep LSTM to learn the temporal structure across the successive time steps. For instance, google CLDN [95] and the Microsoft conversational speech recognition system [91] are based on hybrid architectures.

2.1.2.4. Popular Data Sets

Speech recognition available data sets include TIMIT [73], Switchboard-1 [90], VoxCeleb[74], CHiME5[75], LRS3-TED[76] and Google audio data set [70].

TIMIT is a collaboration between Texas Instruments and MIT (TIMIT) to develop a speech transcription dataset that contains recordings of 630 speakers of the major of American English dialects where each has a ten phonetically rich sentences. Switchboard-1 is a telephone Speech Corpus developed by Texas Instruments in 1990 under DARPA sponsorship. It consists of around 269 hours of speech of about 2,400 two

sided telephone conversation spanning around 543 speakers from the United States. VoxCeleb is a more updated data set with 1,000 celebrities' voice transcriptions. CHiME5 contains multiple speaker natural conversations. LRS3-TED is a visual speech recognition data set that is composed of hundreds of hours of TED talk videos associated with a time aligned subtitles. Google audio set is a collocation of 2 million human labeled 10 seconds sound clips encompassed in 623 audio class

2.2. Architecture Layer

This layer which is shown in Figure 11 can be visualized as a two dimensional layer, the vertical one is considered with the fundamental network architectures which are built conceptually using different structures (i.e. CNN vs RNN) while the horizontal one is considered with the different flavors of networks within the same fundamental architecture (i.e AlexNet[65] vs ResNet[29] in the CNN). The first shall be covered in this section, meanwhile the latter is discussed abstractly except for the CNN networks which shall be discussed in details within the next chapter.

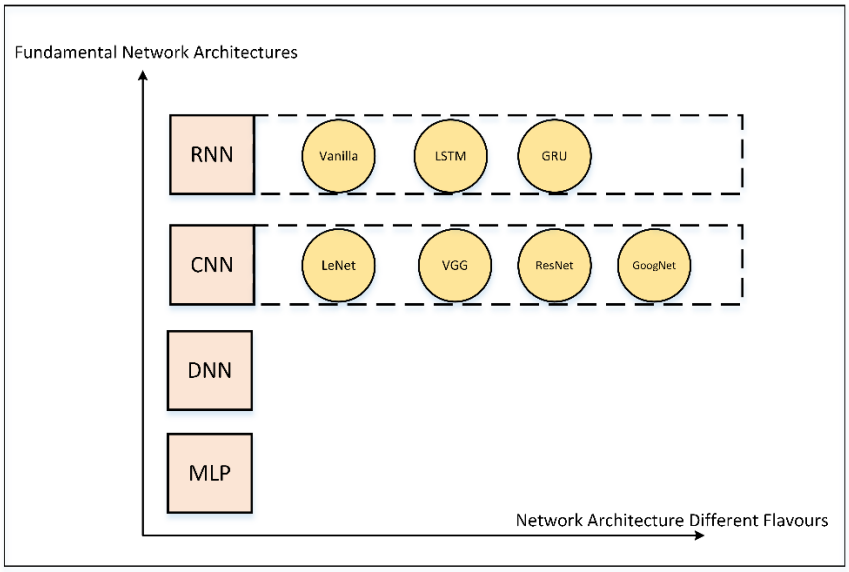

Figure 11 : Architecture Layer two dimensional illustration

2.2.1. Multi-Layer Perceptron

Also referred to as feedforward Networks or fully connected networks. They are based on NN and known for being a basic network in the ML world, however it is considered to be the basis of the DNN.

As shown in Figure 12, similarly to NN it is composed of input layer, one or more few hidden layers and output layer to generate the final network output. The advance form one layer to the another shall imply applying a nonlinear function on a weighted sum computed form all the outputs generated from the previous layer. Thus the layer's connection within this network are described as fully connected layer in which every output from the prior layer contribute in the computation of every neuron in the next layer. The usage of a nonlinear function is essentially required to prevent the whole network form collapsing into a linear transformation function, meanwhile allowing the

learning of complex functions that can be careful in capturing the minute details while suppressing irrelevant variations

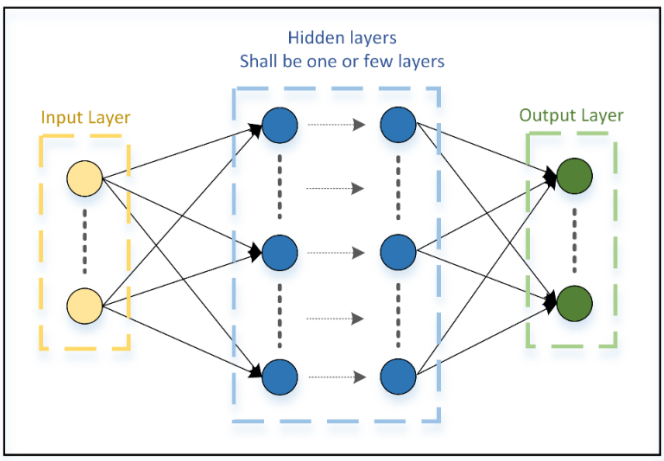

Figure 12 : MLP Abstract network

Usually considered in modelling the non-linear relationship between inputs and outputs with the constraint that they map a fixed input size to fixed size output as well as applying the same input shall always generate the same output regardless how the stream of inputs is fed to the network since this type of network doesn't have any memory effect

2.2.2. Deep Neural Networks

2.2.2.1. General Overview

DNN is a part of the Deep Learning family which is a rich family of multi-layered algorithms comprising NN, graphical models and hierarchical probabilistic models with the supervised and unsupervised feature learning capabilities.

The NN based methods are mainly considered as an extension from MLP where their multi-hidden layers can go beyond few layers to hundred or even thousands layers with billions of neuronal connections to be able to manage the growth rate of the data and tackle the increasing accuracy demand and enhancing its capability to solve the evolving complex problems. For instance, according to [19] Google cat recognizer system has up to 1 billion neuronal connections while this number increases in Baidu Brain to reach 100 billion neuronal connections

Their key feature which helped in their emerging and attracting the wide popularity is their powerful multi-level representation capacity where they automatically extract implicit hierarchical features and patterns from the raw data through nonlinear composition allowing the transformation of the raw input (i.e. pixel of an image) into more abstracted representation to the extent that the representation at one level is transformed into higher abstracted one in the next level , facilitating the amplifying of the required aspects for discrimination while suppressing any irrelevant information. In addition to, distributing the learning across multi-representation levels enhance the network ability to generalize beyond the features that had been learnt through training through the ability to create new combinations of features that might not be available during training. Moreover, the majority of natural signals has a compositional nature

where high level features shall only have extracted through the composition of the lower ones

DNN leap advancement was feasible through the availability of the large annotated data sets that can exhibit the learning capacity of these giant networks to be capable of automatically detect features and patterns without the need of any handcraft support, the dramatic enhancement in the computing capabilities especially GPUs which crossed the threshold of being powerful to handle massive amount of weighted sum calculations in a reasonable time as well as their affordable price, evolution of innovative network architectures that stretched the DNN power such as CNN and RNN and inventing an efficient method to execute the learning techniques especially the Backpropagation.

2.2.2.2. Life cycle phases

Like the any NN, they have two phases across their usage life time: training phase and inference phase

2.2.2.2.1. Training Phase

Training phase is used to determine the network parameters mainly the weight and bias that minimize the network loss function using a well-known data set.

Weights are usually updated using gradient descent which is a hill climbing like optimization process that indicates how the weights shall be adjusted to satisfy the cost function.

Gradient descent is usually implemented using Backpropagation algorithm, a calculus chain rule based algorithm that can derive the partial derivatives of the gradients. Backpropagation operates using the feedforward and backward passes of network as shown in Figure 13. It mainly works by feeding the network with several input samples noted as mini-batch, activating the forward pass, squeezing out the output then computing the derivative of the cost function with respect to weight and bias starting from the output layer gradually to the input layer using the calculus chain rule and the gradient values are then passed backward across the whole network to determine how the loss is affected by each weight and adjust the weights accordingly. This operation is an iterative one where the training sequence is repeated on the whole data set sufficient number of times to ensure the objective function had fallen in a good minimum point. Also, the training procedure is associated with a hyper parameter tuning process that either used to optimize the topology of the network the training configurations. The first is done through ensuring the selection of a sufficient number of layers and number of neurons meanwhile, the latter modifies the weights learning rate and the regularization techniques. Training a DNN nowadays requires a huge data set that may take several days or even weeks to reach the required accuracy as well as the huge computational power and storage needed to build and train these networks.

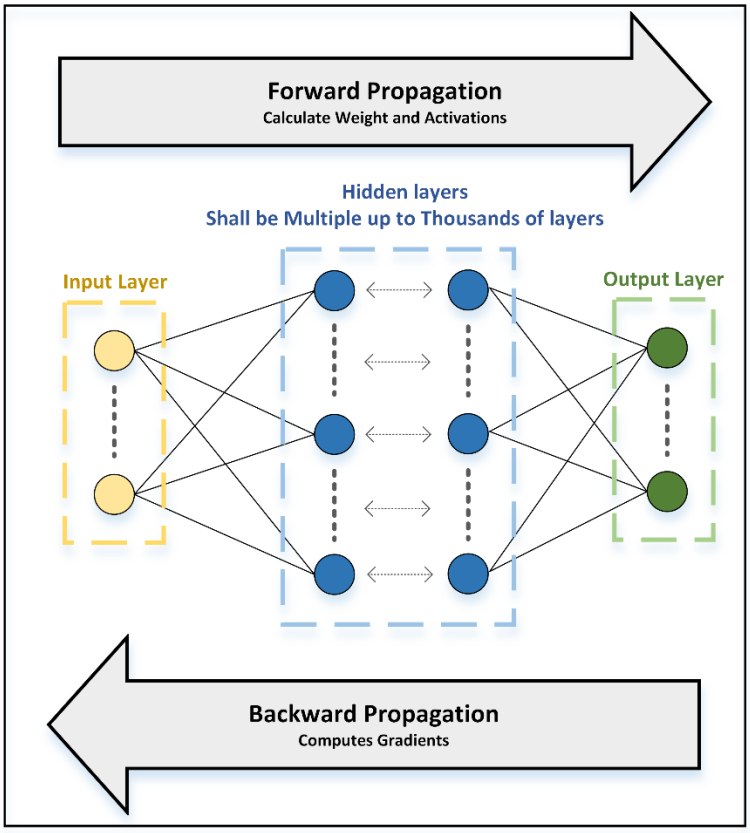

Figure 13 : DNN abstract network forward and backward passes

As the network becomes deeper the vanishing or exploding gradients problems start to become a matter of concern as it may result in a slower training time or falling into a poor local minimum. The vanishing problem arise when the back propagated gradient error is small such that when it reaches the layers close to the input it diminishes, similarly the exploding gradient where the gradient error is increasing exponentially as it propagated back through the network such that when it reaches the layers close to the input it saturates. Historically according to [7], this was partially the reason for directing away from NN towards shallow models (i.e. SVN) where unlike NN they have convex loss objective function that can be efficiently trained to fall within global minima. However, practically local minima are rarely a problem in DNN given that the parameters are carefully initialized as well as the using ReLU as an activation function where an activated neuron has a one constant gradient while clipping any negative values. Moreover, the optimization landscape is packed with large number of saddle points where the gradient is zero and almost of all them are similar for the optimization function, thus it doesn't important which one of them to stuck at.

2.2.2.2.2. Inference Phase

Inference phase is used to run the application in the feedforward pass only of the network using the trained weights. Nowadays, inference may take place using datacenters or edge devices.

2.2.3. Convolutional Neural Networks

2.2.3.1. General Overview

They are NN based networks, mainly considered an extension from DNN that is capable to operate on data that has a temporal or spatial continuity nature. They were inspired from [42] where the visual cortex of a cat was characterized to be sensitive for a small sub-region of the visual field. Admittedly, they are invented on the fact that many natural data are captured in arrays format. For instance, language sequences have one-dimensional format, images and audio spectrograms has two dimensional format and videos has three dimensional format.

2.2.3.2. Key features

The distinct ideas behind CNN are based on its ability to take advantage from the properties and structure of the data nature to introduce concepts like receptive field, feature map, channel pooling and shared weights.

2.2.3.2.1. Receptive field

Receptive field as shown in Figure 14 defines a local sliding window where only a small neighborhood of the input contributes to generate the output meaning that all the inputs within this window at the current slide shall participate in the weighted sum used in the output activation, otherwise the inputs beyond this window their weight shall be set zero. In other words, this can be viewed as if a local connection is created between a spatially nearby subsets of the inputs and the generated output which in return shall reduce the connection within the network compared to a fully connected one leading to a drastic reduction in the CNN number of parameters when compared to a conventional DNN.

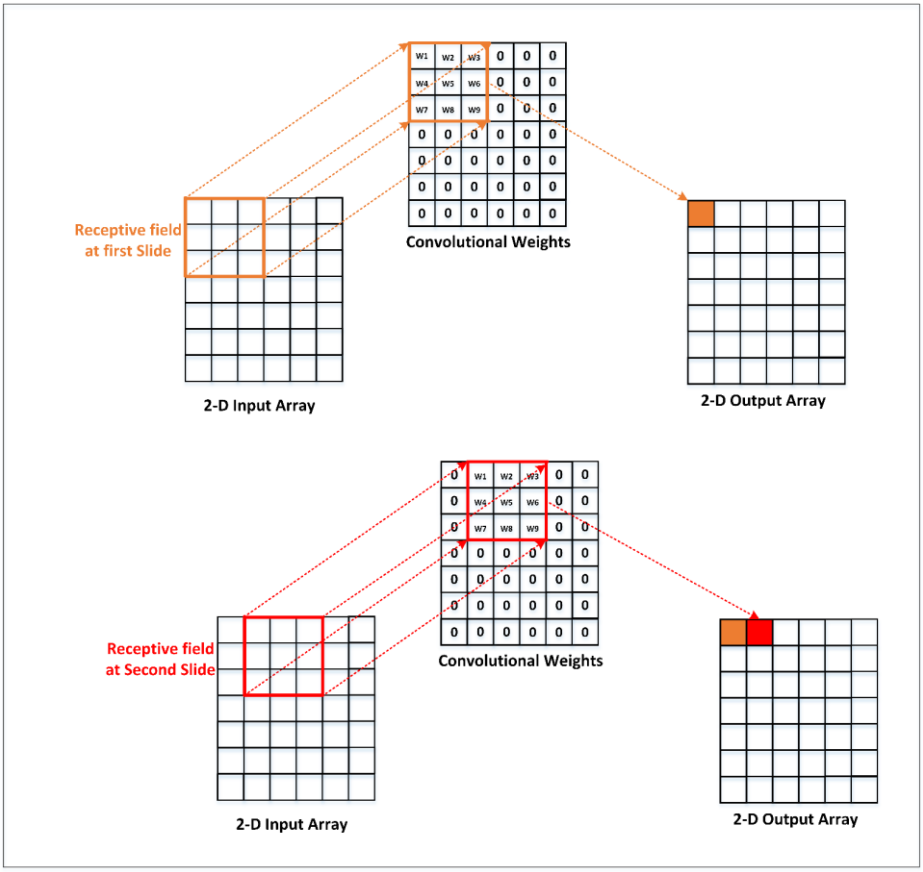

Figure 14 : Receptive field for two sliding windows

2.2.3.2.2. Feature map

Feature map which is shown in Figure 15 defines the interaction between different network layers, where the information is transformed to a higher level of abstraction that preserves the necessary unique features. Mainly, it stacks the data into a two dimensional arrangement noted as channel, where a set of stacked channels forms the feature map. Hence, the feature map shall have a three dimensional arrangement the data height, data width and data number of channels.

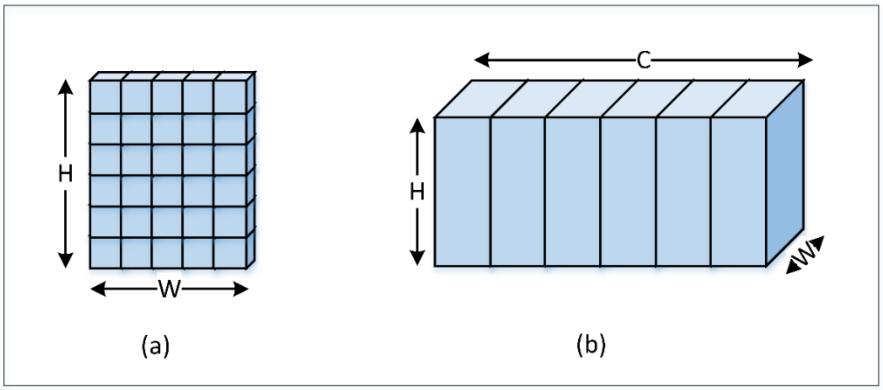

Figure 15 : (a) Feature map with single channel (b) Feature map with C channels

2.2.3.2.3. Channel pooling

Channel pooling which can be viewed in Figure 16 whereas a feature map subsampling technique is applied to aggregate its statistics. Mainly, it is used to merge the similar features within the same channel of the feature map shrinking the feature map dimensions while increasing the robustness of the network and its invariance to small shifts and distortions by detecting the feature representations based on their fine-coarse positions and appearances allowing them to vary a little within the feature map. Moreover, the reduction of the feature map can help in widening the receptive field within the new generated feature map allowing the extraction of larger features from the original feature map. In addition to, reducing the number of computations overhead through diminishing the feature map spatial dimensions.

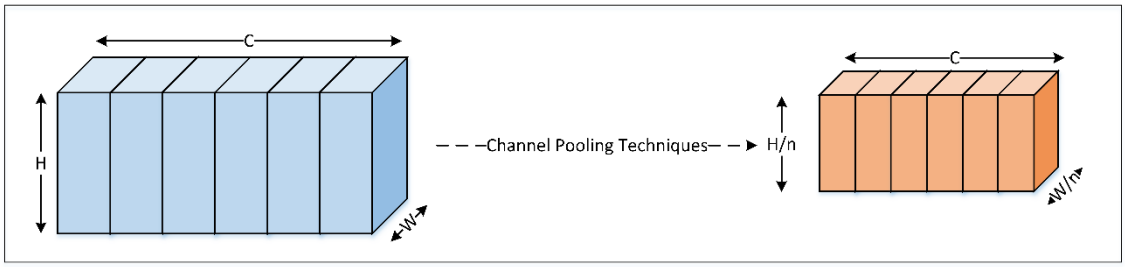

Figure 16 : Feature map before and after channel pooling where n is the pooling scaling value

2.2.3.2.4. Shared Weights

Shared weights as shown in Figure 17 defines the shared parameters of the learnable kernel bank (noted also as filter bank) which is applied on the entire same feature map, where each feature map shall be associated with a unique kernel bank, that is shared across the same feature map but differs ongoing to another one. Sharing weights between different location of the same feature map take advantage that the nature of some data (i.e. images) their local group of values shall be highly correlated and that they are location invariant enhancing the network capability to detect the same pattern at any location within the feature map since they share the same kernel weights. Weight sharing accompanied by channel pooling property confers the CNN with translation invariance property

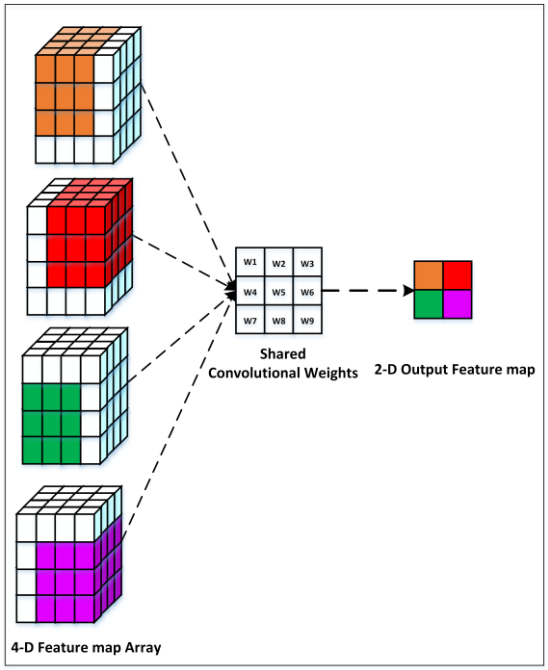

Figure 17 :Feature map with four channels where the same kernel is applied across the entire map to generate an output feature map with single channel

2.2.3.3. Typical CNN Architecture

A typical CNN architecture as shown in Figure 18 is composed of different types of layers mainly the Convolutional layer, pooling layers, fully connected layers and normalization layer where each layer is eligible to generate a feature map to the next layer.

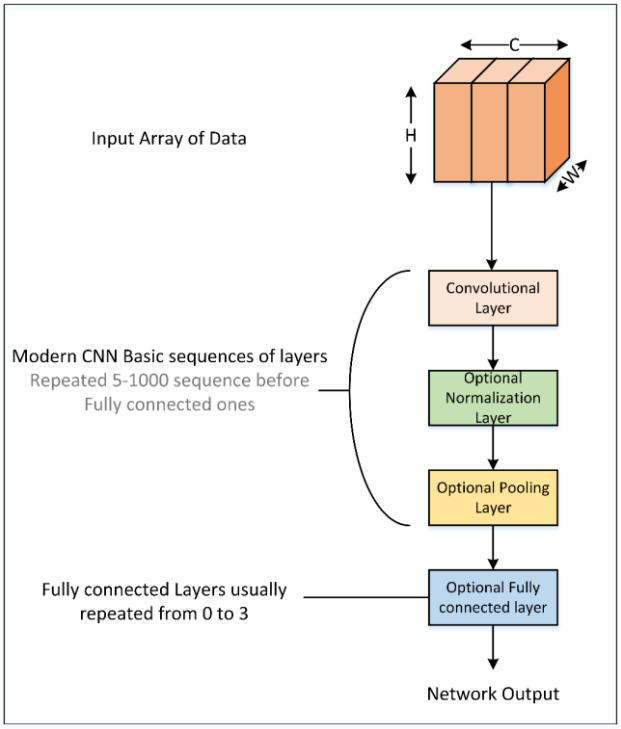

Figure 18 : Typical Modern CNN different layer structure

2.2.3.3.1. Convolutional layer

Convolutional Layer gets its name from the fact that their operation is mathematically a discrete convolution operation (actually a cross correlation one) with emphasis on high dimensional convolution. This is the layer where the dominant number of computations of the CNN takes place. As shown in Figure 19, The input data to this layer is the channel feature map, meanwhile, the learnable kernel bank is stacked according to the required number of channels into a set of two dimensional arrangement keeping in mind that the learnable kernel weights are shared within the same feature map. The kernel bank shall have a three dimensional arrangement: the kernel height, kernel width and the required number of kernel channels. Subsequently, each channel form the channel stack is convolved with a distinct moving kernel channel from the kernel bank. Meaning that, unlike the conventional convolution where the entire input is used to generate one output data, the convolution here is localized through the usage of a regional kernel that scan the feature map in a sliding window liked style such that each shift of the window results in generating a single output data and the full scan shall generate the whole output data. After that the result of every point of this convolution is summed across the whole channels followed by a nonlinear activation function to generate a new feature map for the next layers. Stacking more kernel channels in the kernel bank and convolving them with the input feature map would result in generating more channels in the output feature map. The convolutional layer acts as a feature extractor to identify any local conjunctions and common embedded regional characteristics within the feature map.

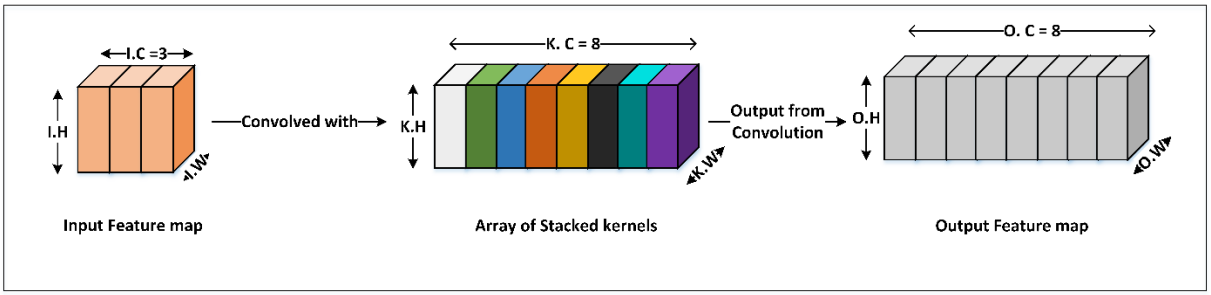

Figure 19 : Example for the convolutional layer where an input feature map with 3

2.2.3.3.2. Pooling layer

Pooling Layer is based on reduction of the spatial dimensions mainly the height and width while keeping the channel dimension as it is. This layer is a computational free one since it has no parameters to learn due to its special operation. Nowadays, applying a maximum or an average pooling is the standard practice. An example is shown in Figure 20 where the stride defines the step window of the non-overlapping blocks associated with separate example for both maximum and average pooling.

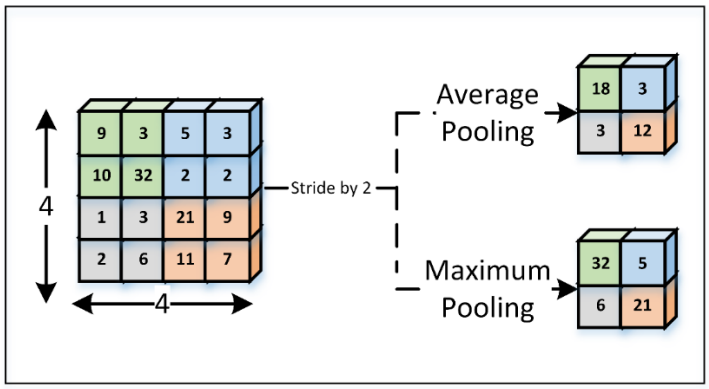

Figure 20 : Feature map with a single channel is reduced through average and maximum pooling with striding by 2

2.2.3.3.3. Fully connected layer

Fully Connected Layer acts as a classifier layer that correlates between the extracted features organized in the feature map and the required logits output of the network assisting in the mapping of the input to the output likelihood category it shall belong to. Recent networks shall have a few of them (one up to three) are appended at the end of the network after the convolutional and pooling layers to perform the classification or the regression objective. In this layer the output activation from the previous layer is connected to every neuron within this layer using an independent weight synaptic, hence losing the weight sharing advantage found in the convolution layer as well as contributing with a reasonable amount from the overall network number of parameters. Consequently, it can be followed by a nonlinear activation function.

2.2.3.3.4. Normalization layer

Normalization layer is responsible to control the feature map statistical distribution by normalizing the input activation such that it has zero mean and unit standard deviation. This is beneficial in terms of speeding up the training by reducing the data space distribution contour, thus reducing the number of iteration. Also, it enhances the achieved accuracy by introducing some noise in the data allowing a better generalization. There are many types of these layer for instance, local contrast normalization (LCN), local response normalization (LRN) and Batch Normalization(BN). Nowadays, BN is the current practice used by ML community given its efficiency and the fact it has minimal computations compared to the convolutional or the fully connected layers

2.2.4. Recurrent Neural Networks

2.2.4.1. General Overview

They are NN based networks that mainly considered an extension from DNN, in which it is capable to handle sequential learning whereas the data has a sequential nature and the application impose sequence to sequence mapping such as language modelling and audio/video description.

2.2.4.2. Key features

Its prominent advantage traits are the memory effect, the ability to work on arbitrary input and output length and finally the weight sharing.

2.2.4.2.1. Memory effect

The memory effect is the ability to track the temporal state of the input by accounting the input history when processing the new ones allowing the dependency of data which is required in sequential data (i.e. speech recognition or language modeling). This effect is constructed through the recurrent connection which creates a loop allowing the information to persist when proceeding from one step to another, so that unbounded amount of information is employed to enhance the accuracy of the prediction. Obviously, unfolding this recurrent can transform RNN to a very deep feedforward network. The Recurrent connections as well as its unfolding can be shown in Figure 21

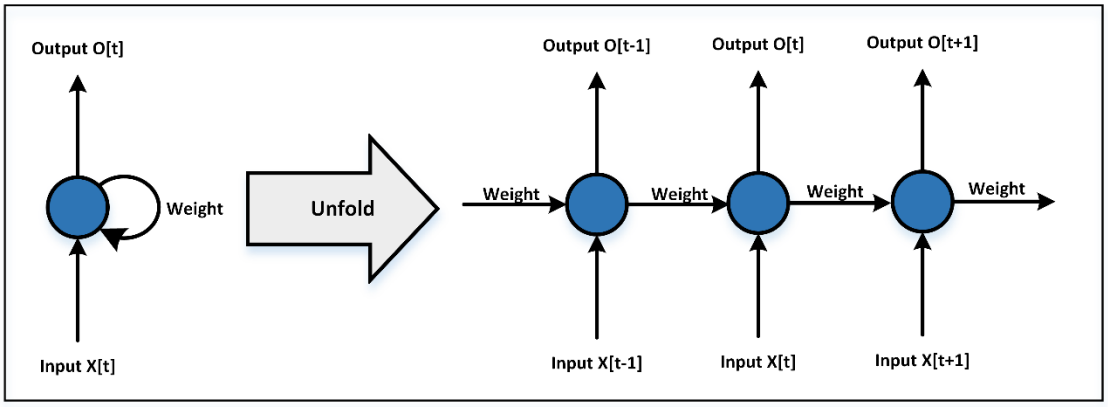

Figure 21 : Recurrent connection and its unfolding equivalence

2.2.4.2.2. Arbitrary input and output length

Arbitrary input and output length is an inherited feature within the sequential data where the sequences length vary across the time. Typical DNN including its CNN flavor fundamentally fixes the dimensions of the input and output limiting their ability to handle such sequences meanwhile RNN can deal with such variation thanks again to their recurrent connections which can execute recurrently for every input within the given sequence. RNN can map the input sequences to the output sequences in many ways depending on the targeted application, for instance as shown in Figure 22 where (a) one to many mapping which is used in image captioning, (b) many to one mapping which is used in sentiment analysis, (c) many to many mapping which is used in machine translation and (d) another many to many mapping which is used in language modelling

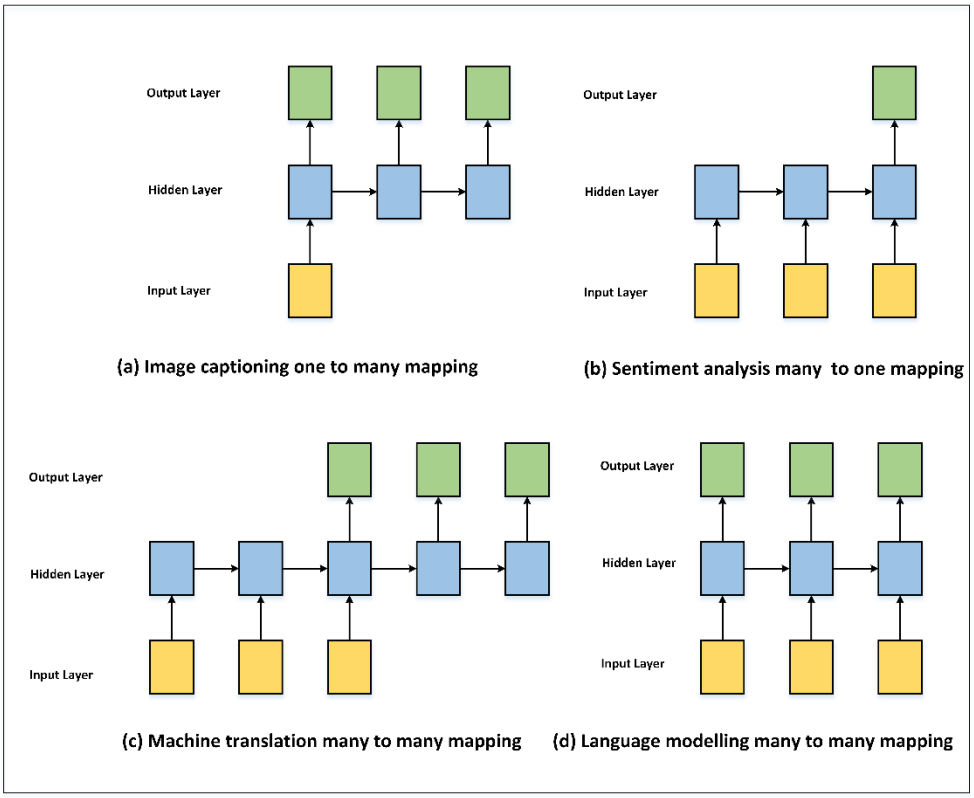

Figure 22 : Different RNN mappings with their target mapping (a) image captioning one to many mapping (b) sentiment analysis many to one mapping (c) machine translation many to many mapping (d) language modelling many to many mapping

2.2.4.2.3. Weight Sharing

Weight sharing within each step across the whole sequences allows network to decouple the arbitrary sequence length and the model structure, meaning that it allows the model to have the number of parameters regardless of the sequence length. As well as, it increases the robustness of the network through being location invariance where the representation features and patterns shall be learnt once regardless at which part of the sequences they appear increasing its ability to generalize well beyond the sequences length that appeared during training experience. Moreover, it drastically reduces the number of learnable parameters compared to having a separate weight for each step

2.2.4.3. RNN training

RNN training has been proved to be problematic since vanishing or exploding gradients issue which was discussed previously amplifies as the recurrent connection would imply the repetition of matrix multiplication resulting in a quick exponential shrink or growth of the magnitude of the gradients. This can be solved similarly to the DNN with carefully initialization of the weights and applying the ReLU as the nonlinear activation function. Moreover, clipping the gradient magnitude is an additional technique applied where an upper and lower thresholds are set, once crossed the gradients are clipped to prevent them from vanishing or exploding

2.2.4.4. RNN State of the art architectures

Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are the most widespread RNN architectures. LSTM can exploit long term dependencies where it preserves the useful information for a longer time delaying its dilution thus enhancing the accuracy on the cost of adding more parameters, longer training time and more computational power. On the other hand, GRU can exploit short term sequence dependencies, requires less parameters, trains faster and reduces computational power compared to LSTM with a drawback that the useful information can be diluted over a short time impacting the accuracy achieved on long sequences.

2.2.4.4.1. LSTM

LSTM in similar to DNN, it consists of a stacked input layer, multiple hidden layers and the output layer, however the building unit is fundamentally different where instead of a normal neuron with a nonlinear activation on the weighted sum, it is modified to include an explicit memory storage to establish the recurrent connection with the help of some framework organizer noted as a gate. This allow the regulation of the flow information in terms of deciding which part of information shall proceed and which shall be forget. An abstract figure of LTSM building unit can be shown in Figure 23 where in addition to the input there is the cell state which represents the memory storage that allows the recurrent connection to be established.

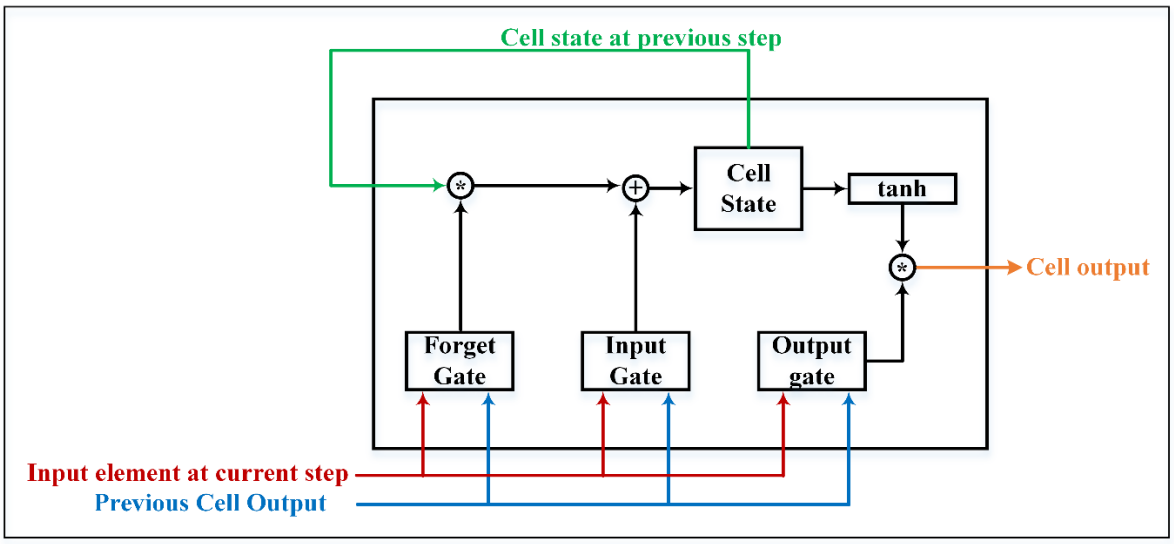

Figure 23 : Abstract LSTM cell

There are three gates that controls the state update as well as generating the output. They can be viewed as a special multiplicative unit. The forget gate defines the amount of memory the cell has to forget which are no longer useful to be stored. This is done by scaling the internal memory state which adaptively can result in the cell forgetting part of its state. The input gate shall define the amount of input that required to be memorized. The output gate defines the amount of information that shall proceed to next cell.

LSTM can have two variants: unidirectional and bidirectional. Unidirectional variant which is shown in Figure 24 considers only the past information during the current execution step. On the hand, Bidirectional variant show in Figure 25 accounts for the past

as well as the future information during executing the current step thus sloping the achieved the accuracy upwards.

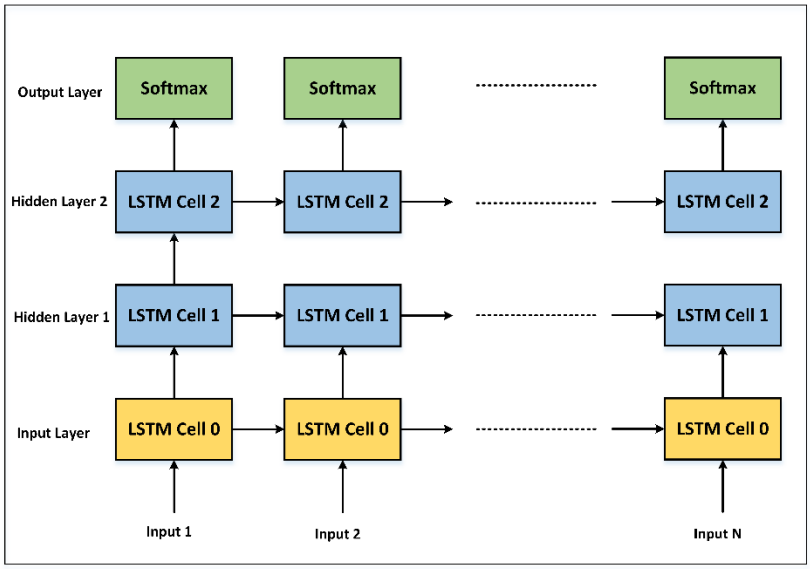

Figure 24 : Unidirectional unfolded RNN example with three LSTM cells, N inputs, two hidden layers and one output layer

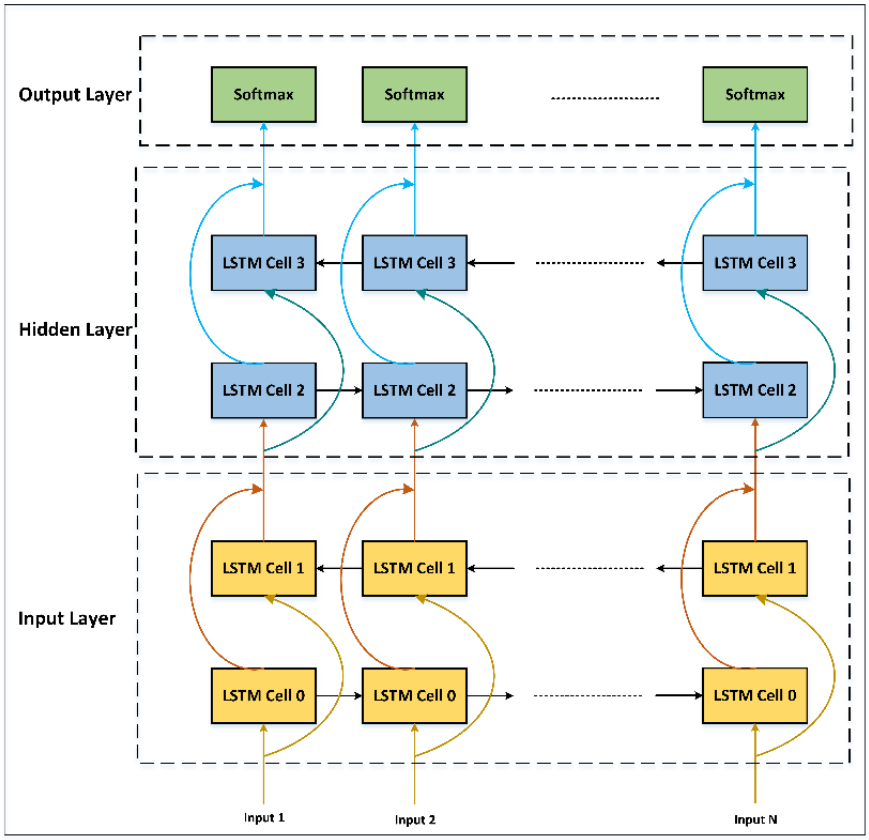

Figure 25 : Bidirectional unfolded RNN example with four LSTM cells, N inputs, one hidden layer and one output layer

2.2.4.4.2. GRU

GRU can be shown in Figure 26 it inherits most of its feature from LSTM but with significant differences. Starting from the removal of the memory state and merging it with output state, followed by merging the forget gate with input gate into a single gate noted as update gate which shall be responsible on passing the amount of information to be stored as well as the amount to be forgot. This allowed the reduction of number of parameters and the structure complexity leading to a more efficient computations and faster training time.

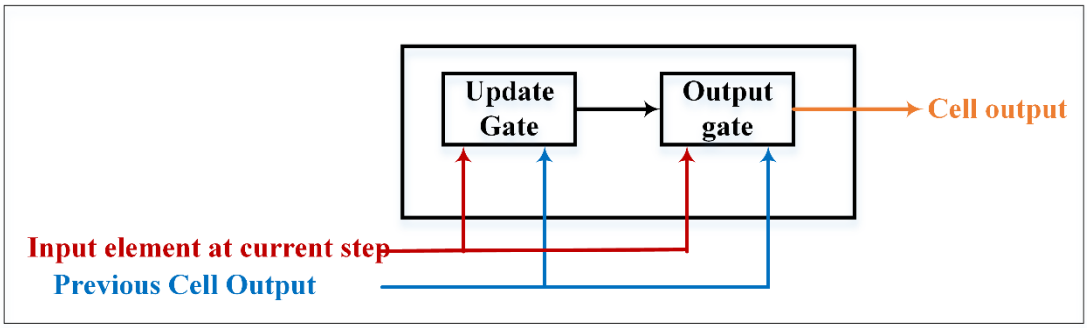

Figure 26 : Abstract GRU cell

2.3. Software Layer

The Software Layer focuses on exploring how to model a network as well as how it shall be implemented.

The Modelling part shall explore different degrees of freedom available to optimize the network enabling its practical deployment in today consumer computing platforms such as mobile platforms and IoT devices. These optimization techniques for instance shall include reducing the network precision, pruning the weights, exploring the network activation statistics and other available ones.

Meanwhile, the implementation part shall show the different tradeoffs between software language approaches. For instance, using a higher level one such as frameworks would accelerate the implementation while a lower level one such as python would enable a full control on each operation.

2.3.1. Network Model

2.3.1.1. General Overview

Earlier DNN approaches had considered their merit of figure to be the accuracy. Thus, they focused on maximizing the accuracy without paying much consideration to other design aspects such as hardware implementation complexity. For instance, the cost of floating point operations, number of parameters required to be stored in the memory and the consumed power to perform the required inference operation. This had led to a more hypothetical networks that are challenging to implement and deploy in nowadays computing platforms.

Recent approaches co-design the DNN models and hardware together leading to the evolving of a new merit of figure that is considered with maximizing the accuracy and

throughput while minimizing the energy and the cost of hardware infrastructure such that it increases its adaption likelihood. However, limiting the available hardware resources would result to a degradation into the achieved accuracy. Thus, the goal has shifted to model a network that matches toady computing infrastructure with the minimum accuracy loss.

To fulfill the aforementioned goal, some techniques were proposed that rely on DNN networks inherent resiliency to insignificant errors. Starting from reducing the network precision where the expensive floating point that requires a complex arithmetic unit and consumes large memory size is replaced with a reduced arithmetic precision representation. Moving to compressing the network itself to get rid of any redundant operations and over parameterized parameters by means of pruning the weights, exploring the activation statistics, low rank factorization and knowledge distillation. In addition to the mathematical transformations techniques where the operation is mathematically reshaped to reduce the number of operations (i.e. multiplication).

2.3.1.2. Reduced precision

2.3.1.2.1. General Overview

Quantization can be defined as mapping the data values from their natural wide set levels to a smaller set of discrete levels. Hence, the quantization process is associated by an additive error and the objective then is to minimize the mapping error between the original levels and the quantized discrete ones.

Precision can be viewed as the number of quantized levels and clearly it is reflected in the number of bits required to map the data to these quantized level (i.e. log2 (number of quantized levels)). Thus, Reduced precision can be referenced to reducing the number of bits that represents the quantization levels.

Reduced precision models focus on transforming the expensive floating point operations which is usually used to obtain the state of art accuracy to a half precision floating one or even to the cheaper fixed point operations which fixes the radix position within the operation. This can be beneficial in terms of relaxing the computation infrastructure and the memory storage requirements. Moreover, optimizing the precision of different data types across the network is considered the distinct computational efficiency advantage of hardware accelerators when compared to the general purpose computing platforms (i.e. CPU and GPU). Furthermore, Reduced precision in difference with compression techniques doesn't encounter any extra steps or computational overhead cost to operate.

However, moving from floating point to a fixed one without reducing the number of bits that represent the quantization levels would result in the same hardware infrastructure cost specifically same area, energy and memory cost. Clearly, the energy and area cost of addition operation using fixed point as well as the memory capacity scales typically in a linear fashion with the number of bits, meanwhile the energy and area cost of a multiplication operation scale approximately in a quadratic manner with the number of bits. Thus, reducing the precision reflected in reducing the number of bits is the key approach for area, energy and memory savings.

One worthy note to mention here, is the recued precision doesn't impact the accuracy if the data distribution is centered around the zero such that the accumulation operation can move in both directions around the zero and preventing its bias towards only one direction. This is usually achievable using normalization techniques

Recent reduced precision approaches focus on reducing the precision of weights rather than activations given that they dominate the memory storage capacity as well as

the intermediate computations. Furthermore, the focus is on the inference phase rather than the training one given that backpropagation algorithm is based on gradients update which can be ill suited to the precision reduction. Actually the gradients and the learning rate are sensitive to the used precision which may cause their vanishing or saturation. Thus, typically a higher precision is required to ensure the network convergence to good minima. Furthermore, intuitive training can be used to compensate the loss in accuracy that may arise from reducing the precision during inference phase where the network can be fine-tuned and re-trained after reducing the precision to improve its accuracy without any extra cost

2.3.1.2.2. Quantization methods

There are two methods to reduce the precision based on how the data is mapped to the quantized levels which are uniform quantization which uses the same quantization levels across the whole data within the network (i.e. all layers, weights and activations) and non-uniform one which uses separate quantization levels within the network (i.e. per layer quantization). The first is simpler in analysis and implementation meanwhile the latter results in a better accuracy.

2.3.1.2.2.1. Uniform quantization

Maps the data with a uniform distance between the quantization level where the floating point representation is mapped to a fixed point one or to the more sophisticated dynamic fixed point representation which allows the fractional part to vary according to the required dynamic range resulting in a less quantization error since the dynamic range of different parts of networks can vary in a different manner. For instance, the dynamic range of the weights and activations can be different depending on their targeted dynamic range which can result in a better overall network accuracy.

Normally general purpose platforms such as CPUs and GPUs can support operations with bit width of 8, 16 and 32 allowing reducing the precision to these values, however the precision required for DNNs can vary in a finer grained manner. For instance, according to [122] the precision values for weights and activations for AlexNet network can vary between 4 to 9 bits with an accuracy loss around 1%. Meanwhile, Intel Flexpoint[114], is an example of a complex dynamic scaling representation. Clearly, unlike the floating point, the exponent is common across all tensors meanwhile it is different from traditional fixed point as the exponent is updated automatically whenever a new tensor is generated using a proposed algorithm noted as AutoFlex. In addition to ESE [116], which applies a uniform quantization approach to reduce the precision of their proposed speech recognition hardware accelerator where they quantized their LSTM network to 12 bits and were able to achieve phone error rate of 20.7% on TMIT corpus in a comparable performance to the floating point architecture that can achieve 20.4% phone error rate

Moreover, there is the binary nets family which can be viewed as an extreme reduced precision model where it reduces the precision aggressively to one bit allowing a distinct transformation into how the operations are executed where the arithmetic operations (i.e. multiply and accumulate) are switched from using multipliers and adders to bit-wise gates instead (i.e. Xnor and AND gates). Starting form BinaryConnect[123] which introduced the binary weights concepts (i.e. -1 and 1) and used these binary weights to transform the multiplication operations to addition and subtraction while allowing the input and the intermediate data to be real. It was able to achieve 61% top-5 accuracy on the ImageNet dataset. Followed by Binarized neural networks [124] which converts the multiplication and addition operations to XOR operations with 50.42% top-5 accuracy

on the ImageNet dataset. Moving to Binary weight nets [125] and XNOR-Nets [125] which modified how the DNN processes the data form using a scale factor multiplication to recover the dynamic range to preserving the floating point operations for the first and last layers. Binary weight nets achieved 79.4% top-5 accuracy on the ImageNet dataset meanwhile XNOR-Nets achieved 69.2% top-5 accuracy. In addition to the HWGQ-Net [126] which increases the activation precision to be 2 bits instead of a single bit while keeping the weights precision as a single bit. It was able to achieve 85.9% top-5 accuracy on the ImageNet dataset. Furthermore, the Ternary weight nets [127] which allows the weights limit to extended to include the 0 as well as the binary weights which requires an additional weight bit representation (i.e. weight to be represented in two bits) and it was able to achieve 86.2% top-5 accuracy on the ImageNet dataset. It was extended in Trained ternary quantization [128] where the weights only are reduced to a binary representation with a different scale values for the positive and negative weights (i.e., − w 1, 0, w 2) while the activation keeps its floating point representation. It was able to achieve 87.2% top-5 accuracy on the ImageNet dataset

2.3.1.2.2.2. Non-uniform quantization

Maps the data with a non-uniform distance through the usage of a mapping function allowing the distance variation between the levels. Recent approaches follow one of three quantization methods; either the log function quantization or the power of two quantization or the learned one.

2.3.1.2.2.2.1. Log function quantization

The mapping function is based on the logarithmic distribution where the weights and activations are distributed equally across different levels and each level is used more efficiently to reduce the quantization error.

For instance, [129], uses a log2 quantization for a VGG-16 network whereas it represents the levels using 4 bits and was able to achieve 85.4% top-5 accuracy on the ImageNet data set, meanwhile [130] introduces the Incremental network quantization which divide the weights into groups, perform an iterative quantization accompanied by re-training to finally reach a 5 bits representation with 92.45% achieved top-5 accuracy on the ImageNet data set.

2.3.1.2.2.2.2. Power of two quantization

The mapping function defines the quantization levels in a power of two fashion. This would allow converting the power hungry frequently used multiplication operations to the hardware friendly shift operation

For instance, [115] quantizes the weights in a power of two fashion enabling the multiplication operation to be executed as a bit shift operation

2.3.1.2.2.2.3. Learned function quantization

Also noted as weight sharing quantization where the mapping function is determined from the data where the function is learnt by means of learning algorithm such as k-means clustering.

Moreover, some weights are forced to share the same value to reduce the number of unique weights within the network. Clearly the weights are grouped using a hashing function or a k-means method. Then each group of weights are assigned to a single value followed by building a mapping table that is usually referred to as a codebook to map each group of weights to its shared value. Accordingly, an index for each group in the codebook is stored to be able to fetch the weight value back.

This method is beneficial to reduce the memory storage cost of the weights as well as the energy required to move the weights from the memory to the computation unit.

An example of this method is Deep compression [131] where it modifies AlexNet to have 256 unique weight value within each convolutional layer and 16 for the fully connected one and it was able to achieve 80.93% top-5 accuracy on the ImageNet data set with 35x reduction in the total network size.

2.3.1.3. Network pruning

2.3.1.3.1. General Overview

In order, to achieve higher accuracy, the network is usually designed with an over parametrized number of weights. This can be viewed as giving the network more parameters to be explored and tuned during the training phase. However, part of these weight parameters ends up to be redundant and can be pruned (i.e. set to zero). Thus, they can be removed without sacrificing the achieved accuracy during the training phase which could result in savings regarding the number of stored weights, the energy required for fetching them and the required number of arithmetic operations required for processing them. This had led to a research area that focuses on pruning the network to remove any ineffectual weights, expanding the sparsity in the weights parameters and reduce the network complexity

Historically, it was first proposed in 1989, through the optimal brain damage technique [132] where it tries to figure out the impact of each weight on the training loss. After that, weights with low impact are removed and the remaining weights are fined tuned. This procedure was repeated until reaching the required reduction in the number of weights with the desired accuracy. However, this approach is impractical to DNN with large size as it would be difficult to estimate the impact of each weight parameter on the training loss.

On contrast, recent search starting from [133] focused on eliminating the neurons with small activity values where the weights are pruned based on the weight magnitude which shall be a simpler and practical technique. Clearly weights with small magnitude are pruned and the rest of weights are retrained to fine tune their values and restore back the loss in the accuracy. Small magnitude values can be though as zero values and can be loosen to include also the near-zero values which encompass more weights and results in more savings without impacting the accuracy. For instance, in [133] the AlexNet number of weights were reduced nine times while maintaining the same accuracy.

2.3.1.3.2. Area of focus

Advances in the network pruning focuses on two areas the first is how efficiently store the sparse weight after pruning which shall need to a compression format to open the benefits of pruning these weights and how to structure the pruning to allow their processing on general computing platforms (i.e. CPU and GPU) without the need for any custom hardware.

2.3.1.3.2.1. Storing Sparse weights

Compressing the sparse weights shall consider how DNN process these weights through the matrix vector multiplication which is one of the fundamental operations within network. There are two compressing format to be applied either the compressed sparse row format or the column one. Compressed sparse row format when used during the matrix vector multiplication as shown in Figure **27** requires the input vector to be read multiple times while each output element is generated once at a time. Meanwhile,

compressed sparse column format as shown in Figure **28** requires only the input vector to be read once while each output element is updated several times before generating the final one. Compressed sparse column format is more effective than row one as it provides an overall lower memory bandwidth given the fact the number of filters within a DNN is not significantly larger than the number of weights encapsulated within these filters, thus updating the output elements serval times is cheaper than reading the whole input vector the same number of times.

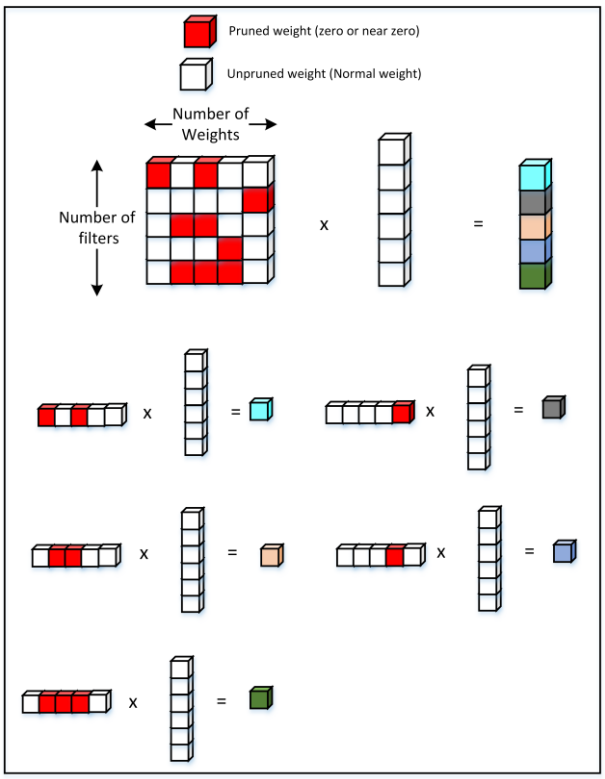

Figure 27 : Compressed sparse row format during matrix multiplication

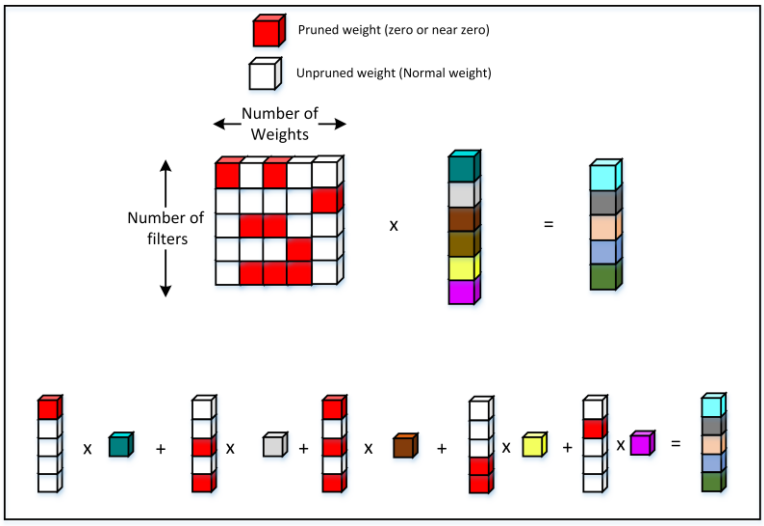

Figure 28 : Compressed sparse column format during matrix multiplication

2.3.1.3.2.2. Structured pruning

Structured pruning can be viewed as a coarse grained pruning where in contrast to the fine grained pruning where individual weights are pruned based on their magnitude, a group of weights are pruned together based on a define criteria which may be a filter entire row or column, a filter channel, a neighboring weights in a filter or the filter itself. Grouping weights together would be beneficial to decrease the cost of locating of non-zero weights which would facilitate compressing the sparse weights and enable their parallel processing using the existing general computing platforms without the need for any customization. However, grouping large weights together would result in an increasing accuracy loss which requires more fine tuning and carefully choosing the grouping criteria

Applying this optimization method is used in EIE [43] which is a hardware accelerator that uses the compressed column format to exploit the weights sparsity. In addition to ESE [116] which is a LSTM hardware accelerator that prunes the unnecessary weights based on an empirical pruning threshold as well as introducing a load balance aware pruning method to increase the hardware utilization through balancing the non-zero weights distribution among all the parallel processing units. Also, Cnvultin[32] which allows dynamically skipping neuron computations if they are below a pre-specified, per-layer threshold.

2.3.1.4. Activation statistics

Recently, there are many work on exploiting the generated content in the hidden layers within the DNN networks with a focus on searching for the abundant sparsity (i.e. existence of zero values within the intermediate data) in aim to get advantage of these sparsity by means of compression to reduce the number of computations which shall result in area savings as well as reducing the energy expensive access to the off-chip DRAM.

Currently, ReLU is the main nonlinear activation function used within the state of art networks due to its efficiency in generalizing the network as well as its simplicity. ReLU as shown in Figure **29** set any negative values output from the neuron to zero. This had led to generation of a large amount of zeros within the hidden layers and these zeros are considered to be an intrinsic property of using the ReLU function. For instance, according to [4] the feature map within the hidden layer of AlexNet can have sparsity between 19% up to 63% depending on the layer.

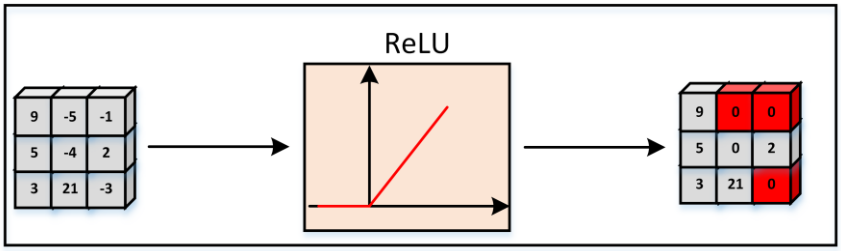

Figure 29 : ReLU function

These zeros generated form the activation can further be explored when designing a network to make an energy efficient network without any performance impact as they don't contribute to the final output of the network and can be optimized, whereas the

computations had been transformed to a sparse matrix multiplication which shall require fewer operations when compared to a dense one meanwhile the memory access can also be safely bypassed given its predetermined it is going to fetch a zero value.

Applying this optimization method is used in LRADNN[21] which estimates the polarity of the inputs going to the neuron and hence, based on this polarity it can disable some of the multiplication operations which led to a reduction in number of arithmetic operations without much accuracy impact. Also, SparseNN[23] is another example which adds a prediction phase to the network which involves the usage of a predictor noted as straight through estimator that has a lightweight computation complexity to be able to determine whether a zero exist in the activation or not. In addition to Eyeriss[48] which uses a compression technique based on an encoding scheme noted as RLC that exploits the zeros within the feature map to skip any unnecessary computations as well as saves any useless DRAM access. Furthermore, Cnvlutin[32] which introduces the Zero Free Neuron Array Format as the compression technique to eliminate the zero activation computations

2.3.1.5. Low rank factorization

Given how the CNN is advancing, more efforts are focused on optimizing the convolution operations which contribute to the bulk of CNN computations. Low rank factorization is a technique that applies matrix decomposition in order to estimate the informative parameters within a CNN. The basic idea is to view the convolutional kernel as a four dimensional matrix where there are a lot of redundant weights. This redundancy can be removed through decomposing this large matrix into smaller ones. To have even more efficient computations, the decomposition is followed by another compression step through approximating these smaller matrices by means of low rank approximation. A demonstration for this method can be found at [138], where Canonical Polyadic (CP) decomposition accompanied by low rank approximation was used and was able to achieve a 4.5x speedup for the second layer of AlexNet with 1% accuracy drop.

2.3.1.6. Knowledge distillation

One way to increase the achieved accuracy is to use network ensembles where multiple network run in parallel but with different configurations (i.e. weight initialization) then average their predictions to get a better accuracy when compared to running a single network. However, this shall increase the required computational complexity. To get a better tradeoff between the accuracy and the computational cost, knowledge distillation is used.

Knowledge distillation can be viewed as a teacher student model where a complex network or an ensemble of networks are defined to be the teacher that is used to bootstrap the accuracy of an architecturally different network that is more compact and shallower that is defined to be the student. This is done by transferring the knowledge learned by the teacher network to the student one in an aim that the student network when trained it shall be able to mimic and reproduce the same output of the teacher or even a better one that would not be achievable if the student was trained directly on the same data set. This shall incorporate defining the loss function of the student during the training to be learning the class distributions output from a softmax layer. For instance, according to [134] using knowledge distillation helped to improve the speech recognition of a student network by 2% which allowed it to be competitive to the teacher which is composed of an ensemble of ten networks.

The way the knowledge distillation works is shown in Figure **30** where the target of the student network is to learn the class scores of the teacher (which may be an ensemble of networks). Class scores are used as the target rather than the class probability as the softmax layer eliminates the small scores by pushing their probability towards 0. However, if a softened softmax is used where the small scores are preserved and a smoother probability distribution can be generated then the class probabilities can be used as a target. Overall, the training objective is to minimize the squared difference between the class scores generated from the student and the target.

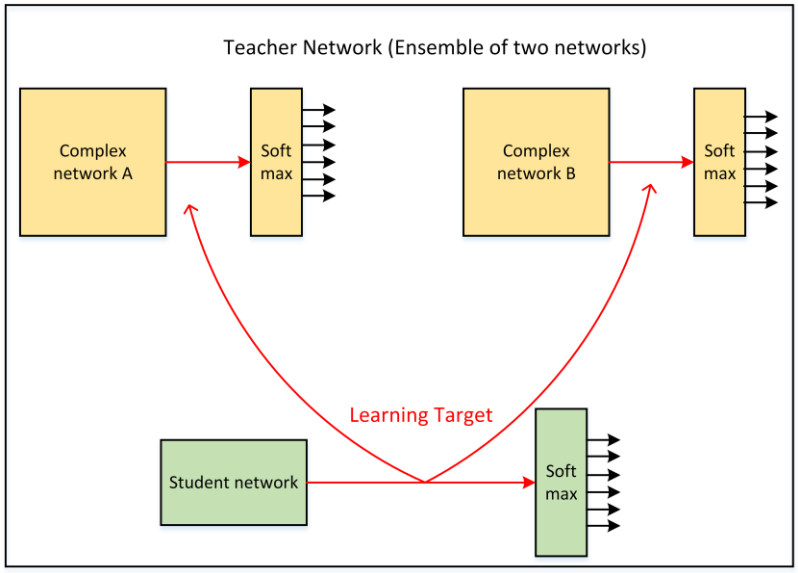

Figure 30 : Knowledge distillation overview

2.3.1.7. Mathematical transformations

Several mathematical transformations are used especially in the CNN to either reduce the required number of multiplications while maintaining the bitwise accuracy or accelerate the execution of the multiplication operation. These types of transformations are targeting the convolution operation where another mapping function is proposed instead of the multiplication based mapping or the convolution is restructured in another accelerated form. This includes Fast Fourier Transform [135], Winograd's algorithm [136], Strassen's algorithm [137] and Structural matrix using relaxed Toeplitz form [4].

2.3.1.7.1. Fast Fourier Transform

Fast Fourier Transform is used to reduce the number of multiplication where the convolution operation is done as a direct multiplication in the frequency domain. As shown in Figure **31** the input feature map and filter are transformed in the frequency domain, multiplied together and then inverse FFT is applied on the result to generate the output feature map in the spatial domain. FFT is usually used with larger filter sizes (i.e. 5x5).

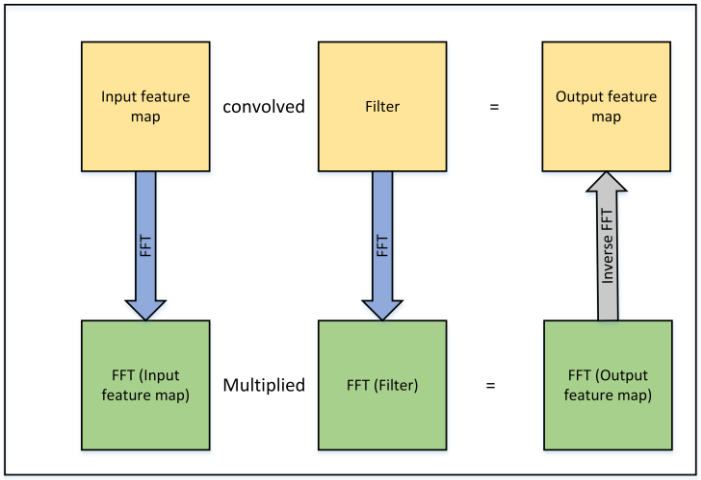

Figure 31 : FFT mathematical transformation

2.3.1.7.2. Winograd's algorithm

Winograd's algorithm applies a transformation for the input feature map and the filter to generate a tile of elements in the output feature map together such that it gets benefit from the structural similarity among them. This help to reduce the required number of multiplication given it generates a tile of output elements at each step. It is usually used in smaller filter such 3x3 where according to [4] it was able to reduce the number of multiplication by 2.25x.

2.3.1.7.3. Strassen's algorithm

Strassen's algorithm reduces the number of multiplication through the rearrangement of the matrix multiplication in a recursive manner. However, it suffers from occasional numerical stability as well as more storage requirements.

2.3.1.7.4. Structural matrix

Structural matrix using a relaxed Toeplitz form as shown in Figure **32** is used to speed up the matrix multiplication by extending the feature map with redundant elements to allow its parallelization. However, this shall come with an inefficient increase in the storage cost and adding extra complexity to the access memory patterns

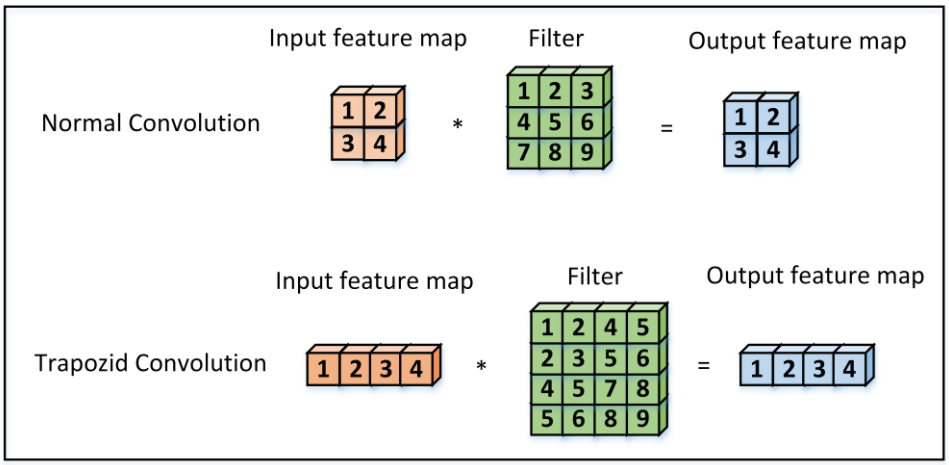

Figure 32 : Structural matrix using a relaxed Toeplitz form

2.3.2. Network Implementation

2.3.2.1. General Overview

There are many approaches to implement a network; Starting from using a low level language such python, matlab and CuDNN until using a high level framework such caffe, tensorflow and keras.

While low level languages provide a full control on the implementation where it is feasible to customize the network operations and apply any optimization method (i.e. quantization), it consumes much time to develop the network from scratch given there are no ready plug and play DNN functions. Also, the network execution time is dependent on the code quality which in return reflects the experience of the code owner leading to more hassles in the implementation part rather than the architecture part.

Meanwhile, high level framework provides open source DNN libraries that implement the common training and inference operations which ease the network development, enable sharing the trained networks, leverage the accumulated experience among the whole ML community. However, customizing an operation would require modifying the open source code of the provided libraries which puts a barrier that consumes a lot of efforts to establish new ideas

2.3.2.2. Low level languages

2.3.2.2.1. Python

An open source general purpose programming language which is built on a collection of generic built in libraries. It is widely used in web applications, mathematical scripting as well as being popular in ML applications where most of the higher level frameworks are built on the top of it.

2.3.2.2.2. Matlab

A commercial programming language that provides a deep learning toolbox facilitating the optimization of the deep learning functions. It is also capable to automatically convert the written code to C++ or RTL code

2.3.2.2.3. CuDNN

The NVIDIA CUDA Deep Neural Network library (cuDNN) is a NVIDIA GPU based accelerated library of primitives for deep neural networks. It provides highly optimized implementations for commonly used DNN functions.

2.3.2.3. High level framework

2.3.2.3.1. Caffe

Convolutional Architecture for Fast Feature Embedding was developed by university of California Berkeley as an open source deep learning framework that can be viewed as a cross platform that supports C/C++, python and matlab. It provides an implementation that can run on both CPU and GPU.

2.3.2.3.2. Tensor flow

An open source framework developed by Google Brain Team and can support C++ and python. It is computation flow can be expressed in a single dataflow graph that

manages all the tensor operations. It provides an implementation that can run on both CPU and GPU

2.3.2.3.3. Torch

An open source framework developed by Facebook and New York university and can support C++ and Java. It provides an implementation that can run on both CPU and GPU. It is no longer under active development.

2.3.2.3.4. Pytorch

An open source framework developed by Facebook's AI research group as a successor for torch and can support C and python. It integrates acceleration libraries such as IntelMKL and NVIDIA (cuDNN, NCCL). It supports a technique noted as reverse mode auto differentiation which all to change the way a network operates with small effort rather than starting to build it from scratch. It provides an implementation that can run on both CPU and GPU.

2.3.2.3.5. Theano

An open source python library developed by University of Montreal. Theano starts performing computations by optimizing the selection of computations, translates them into other languages such as C++ or CUDA and then compiles them into Python modules in an efficient way on CPUs or GPUs. It provides an implementation that can run on both CPU and GPU and it is No longer under active development.

2.3.2.3.6. CNTK

Microsoft Cognitive Toolkit (CNTK) is an Open source deep learning framework developed by Microsoft Research and supports python, C++ and C#. It converts any function to a directed graph where each leaf node consists of an input value or learning parameter, and other nodes represent a matrix operation upon their children. It provides an implementation that can run on both CPU and GPU.

2.3.2.3.7. Keras

An open source framework founded by Google engineer Chollet as a part of research project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) and it can support python. It can be viewed as a higher level library that can run over Tensorflow, Theano and CNTK to unify the development experience and allows faster development. It provides an implementation that can run on both CPU and GPU.

Chapter 3 : Convolutional Neural Network Architectures Review

During the last several decades, many CNN architectures had been developed that differ in terms of number of layers, layer shapes, layer associated parameters (i.e. filter size, number of channels) and how the layers are connected to each other to allow the propagation of feature maps.

Most of the recent architectures were driven by the ImageNet competition [64] where most of them had competed and the innovative ones had won it. ImageNet competition is a tourney with many different tracks.

One of the tracks that remarks the breakthrough of the CNN approach is the image classification. Clearly, before the CNN paradigm the error rate achieved was around 25% however starting from 2012 when AlexNet[65] was introduce by a group from Toronto university were they applied the CNN accompanied by the usage of GPUs for training and successfully dropped the error rate to 16% had marked the start of shift from traditional approaches towards the CNN based approach.

Over the years starting from 2012 as shown in Figure 33, the CNN had continued to improve the error rate in the ImageNet challenge with a significant milestone at 2015 when the ResNet[29] had been introduced whereas it was able to suppress the human level accuracy. Furthermore, from [64] the entrants in the ImageNet challenges that are using GPUs had increased from four entrants only at 2012 when AlexNet was used to 110 entrants at 2014 indicating the domination of the CNN approach.

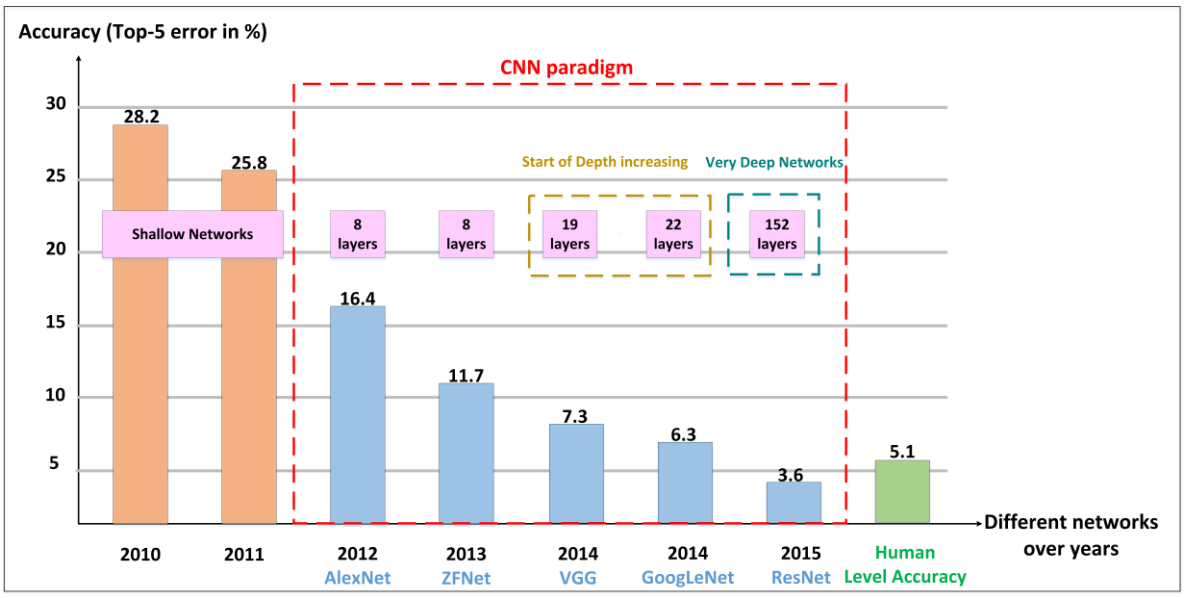

Figure 33 : ImageNet top-5 error accuracy versus different networks progress over years

In this chapter, different CNN are explored starting from LeNet-5[64] until the ResNet[29].

3.1. LeNet-5

LeNet-5[64] was introduced in 1989 as one of the first CNN that was designed for the digit classification task on the MNIST grayscale images [62]. Hand written digit recognition was widely used at that time by ATMs for digit recognition on checks enabling the first commercial use of the CNN through LeNet-5 deployment in ATMs to automatically identify the check deposit digits.

As shown in Figure 34, it is composed of two convolutional layers, two average pooling layers and two fully connected ones. The convolutional layer is based on kernel of 5x5 size where six of them are used in the first layer while 16 are used in the second one. After each convolutional layer a sigmoid function is applied as the nonlinear transformation function followed by 2x2 average pooling layer.

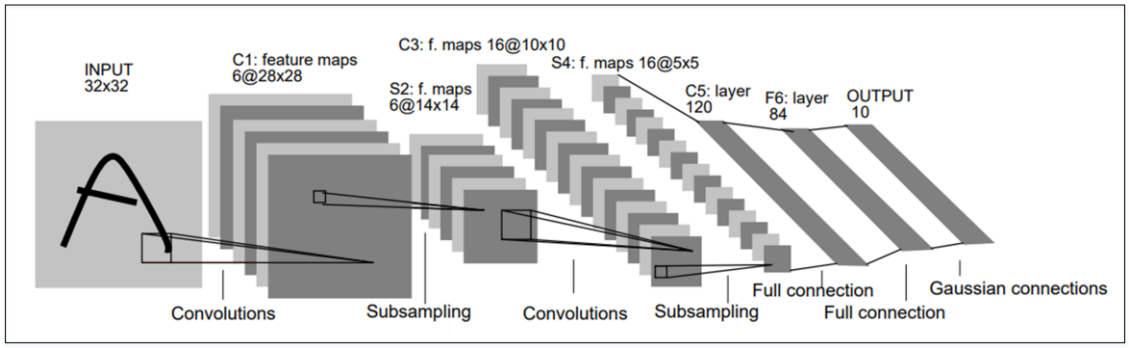

Figure 34 : LeNet-5 architecture from [64]

In general, it had 60,000 weight parameter and was able to achieve to 99.05% accuracy on the MNIST data set

3.2. AlexNet

AlexNet [65] was introduced in 2012 and the first CNN based network to win the image classification track within the ImageNet Challenge.

As shown in Figure 35, it is composed of five convolutional layers, three maximum pooling layer and three fully connected ones. Each convolutional layer may have kernels of 3x3 to 11x11 size with number of kernels varying from 96 to 384 and from three to 256 generated channels depending on the location of the convolutional layer within the network. After each convolutional layer a ReLU for the first time in a CNN is applied as the nonlinear transformation function and the first, second and fifth convolutional layer are followed by a 3x3 maximum pooling.

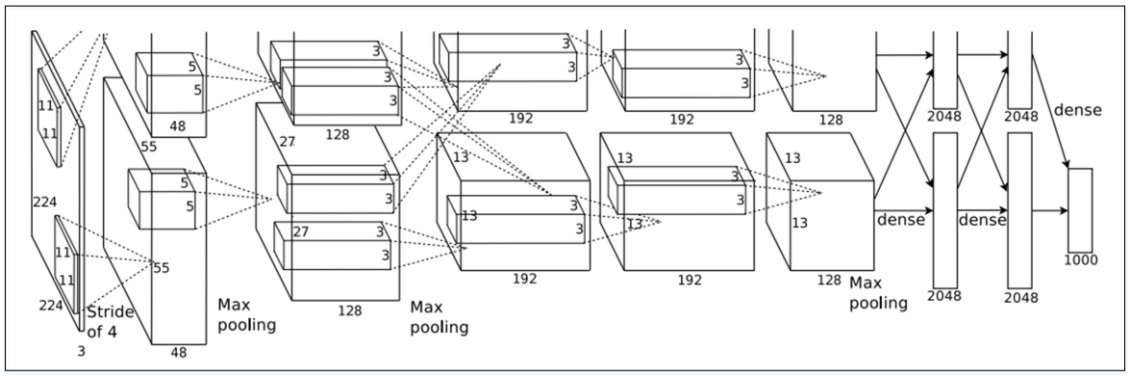

Figure 35 : AlexNet architecture from [65]

The key differences between AlexNet and the LeNEt-5 are the increased number of weights, the kernels varying size and the usage of the LRN as a normalization technique after the first and second convolutional layers.

In general, it has 61 million weight parameters and was able to achieve 16.4% top-5 error on the ImageNet data set.

3.3. ZFNet

ZFNet[104] was introduced in 2013 and was the winner of image classification track within ImageNet challenge.

It is a refinement version from AlexNet where the 11x11 kernels are replaced by 7x7 ones and the number of activation kernels were changed to 512 or 1024 depending on the location of the convolutional layer.

It has the same AlexNet structure with five convolutional layers, three maximum pooling layers and three fully connected ones.

In general, it was able to achieve 11.2% top-5 error on the ImageNet data set.

3.4. Overfeat

Overfeat [96] was introduced in 2013 and was the winner of the object detection track in the ImageNet challenge.

It follows AlexNet in the structure with five convolutional layers, three maximum pooling layers and three fully connected ones.

The main difference is that number of kernels are varied up to 1024 within the convolutional layers based on the layer location within the network.

In general, it has 146 million weight parameters and was able to achieve 14.2% top-5 error on the ImageNet data set.

3.5. VGG

VGG [97] was introduced in 2014 and was the winner of the object detection track in the ImageNet challenge as well as the first runner up of image classification track.

It was one of the first attempts to explore the depth aspect of the CNN. To tradeoff between going deeper and the exponential growth of the number of weights parameters,

it fixes all the kernels size within the network to 3x3 size which has fewer weights parameters compared to larger ones, meanwhile these larger kernels can be built using multiples of the smaller kernels. Decomposing larger kernels into a stack of smaller ones had shown to be fruitful from many aspects; first it attains the same effective respective field of the larger kernels for instance as shown in Figure 36 where a 5x5 kernel can have the same effective receptive field of two stacked 3x3 kernels, second it incorporates multiple apply of the nonlinear transformation function allowing the classification function to be more discriminative , again for instance a 5x5 kernel shall be followed by applying a single nonlinear function ,meanwhile applying the nonlinear function can follow each kernel from the two stacked 3x3 kernels and finally it decreases the required learnable parameters , back for instance to the 5x5 kernel which shall have 25 weight parameters per channel while the two stacked 3x3 kernels shall have 18 only.

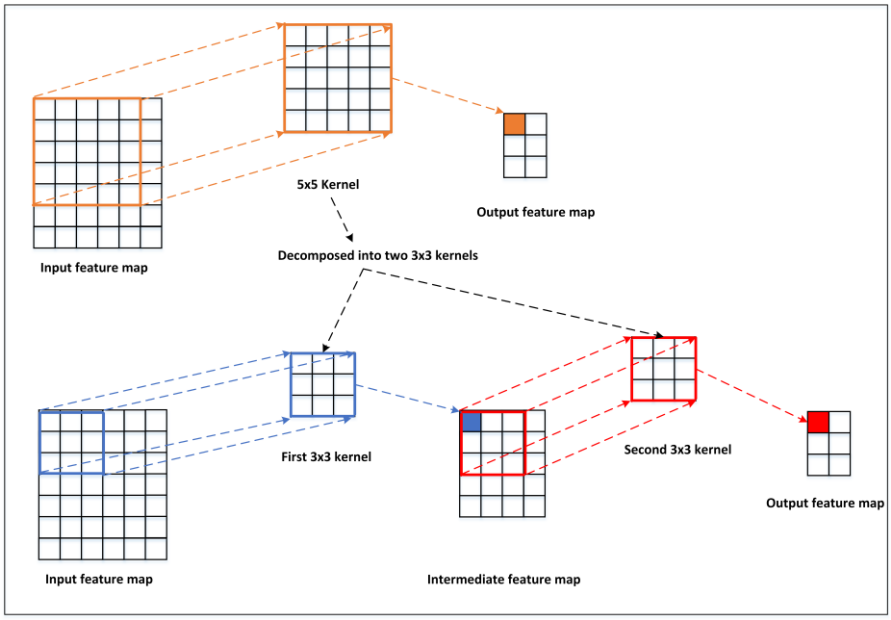

Figure 36 : 5x5 kernel decomposed into two 3x3 kernels

The VGG network shall have a generic structure where it keeps the kernel size fixed at 3x3 while gradually increasing the depth of the network by stacking more convolutional layers, actually as the network goes deeper the generated feature map within each layer is modified through a fixed fashion whereas the number of kernels applied that shall represent the number of generated channels is doubled while the generated height and width dimensions is halved. In general, VGG has three popular variants VGG-11, VGG-16 and VGG-19.

VGG-11 as shown in Figure 37(a) is composed of eight convolutional layers, five maximum pooling layers and three fully connected layers with total 133 million weight parameter and top-5 error of 10.4% on the ImageNet data set. VGG-16 as shown in Figure 37(b) is composed of 13 convolutional layers, five maximum pooling layers and three fully connected layers with total 138 million weight parameter and top-5 error of 7.4% on the ImageNet data set. VGG-19 as shown in Figure 37(c) is composed of 16 convolutional layers, five maximum pooling layers and three fully connected layers with total 144 million weight parameter and top-5 error of 7.3% on the ImageNet data set.

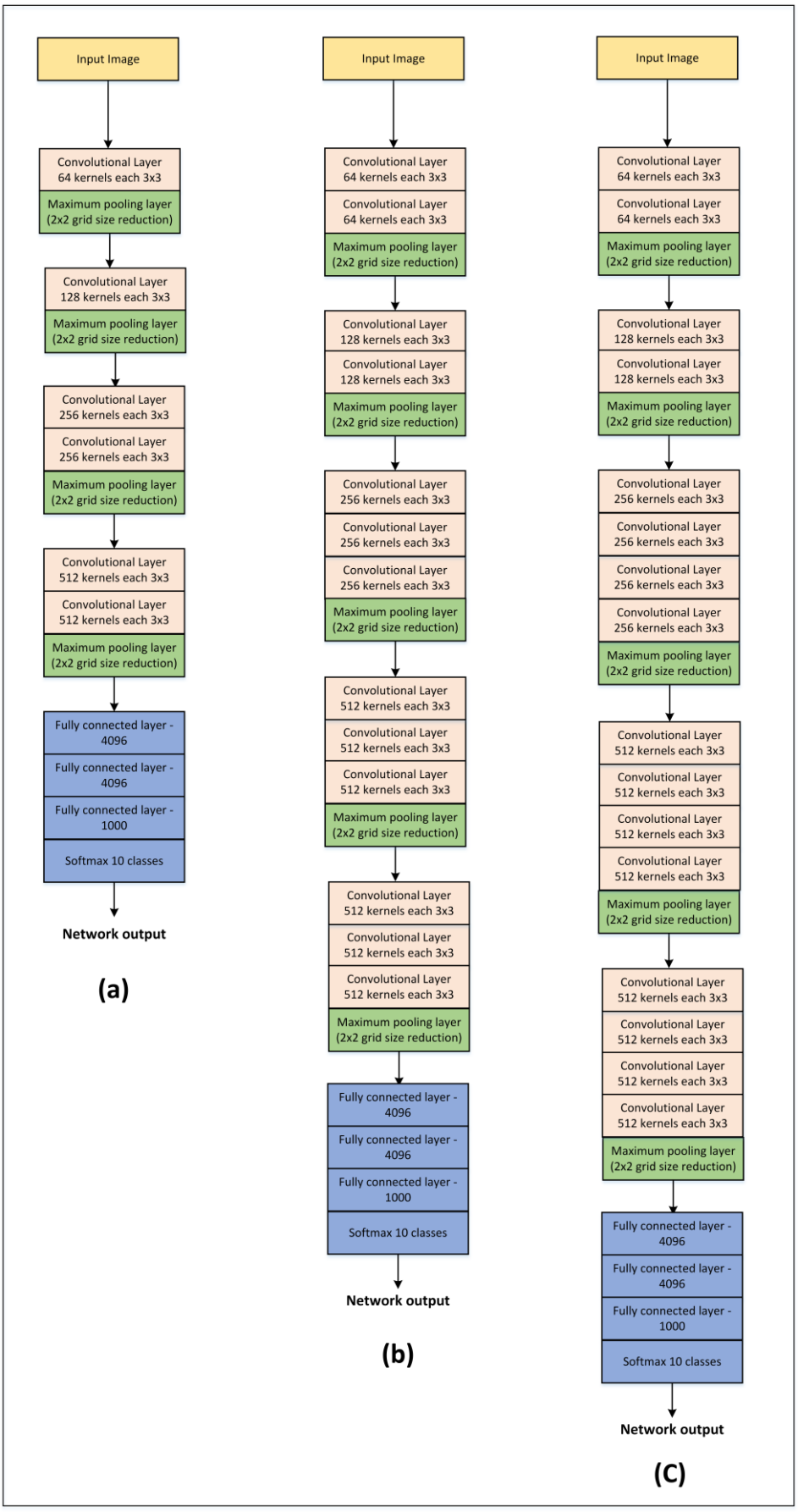

Figure 37 : (a) VGG-11 (b) VGG-16 (c) VGG-19

3.6. NiN

Network in Network (NiN) [105] was introduced in 2014 and didn't participate in the ImageNet challenge, however it is considered the precursor for dimension reduction of the inception module used in GoogLeNet network and the bottleneck module used in the ResNet networks.

It introduced the Mlpconv layer (Multilayer perceptron convolutional layer) where it replaced the linear convolutional kernel and its subsequent nonlinear activation function by a micro multilayer perceptron. The feature map then can be generated by sliding this layer over the input in a similar manner to the normal convolutional layer but with a multilayer perceptron way of computation. The intuition is that if a fully connected layer is applied at each point within the feature map (each height and width) and the weights of this layer is tied across each spatial location then this would be analogous to utilizing a 1x1 convolutional kernel. 1x1 kernels are beneficial in terms of preserving the spatial dimensions (height and width) of the feature map while reducing the depth (channels) to lower dimension (i.e. as if it is generating a combination of feature maps)

3.7. GoogLeNet

Also referred to as Inception [98] that was introduced in 2014 and was the winner of image classification track within ImageNet challenge. Since its introduction it was followed by three versions [99], [100] and [101].

3.7.1. First version

The first version introduced the inception module and started to go deeper with the number of the layers within the network.

The motive behind the inception module is to improve the utilization of the computation resources through moving fundamentally from a fully connected architecture to a sparsely connected one. The idea behind that, if the data set probability distribution can be represented by a large sparse network, then the optimal network topology can be constructed layer by layer through analyzing the correlation statistics of the activations of the last layer and clustering neurons with highly correlated outputs. To illustrate more, assume the neurons in the earlier layers close to the input shall correspond to some regions in the input image where highly correlated ones would mean that they are concentrating on the same local region and can be clustered. Moreover, some of these clusters may end up concentrating on a single region and can be covered in the next layer through a 1x1 convolution kernel. Similarly, there would be spatially spread out clusters that can be covered using convolutions over large patches which can be approached using higher order convolution kernels such as 3x3 and 5x5 kernels. Hence, the optimal local sparse architecture can be approximated and constructed through the combination of all those kernels where all their outputs are concatenated into a single output forming the feature map for the next layer.

Thus the naïve inception module as shown in Figure 38 unlike the proceeded networks has parallel structured connections within the same layer instead of a single direct connection whereas different kernels size (mainly 1x1, 3x3, and 5x5 kernels) and a maximum pooling layer are concatenated together.

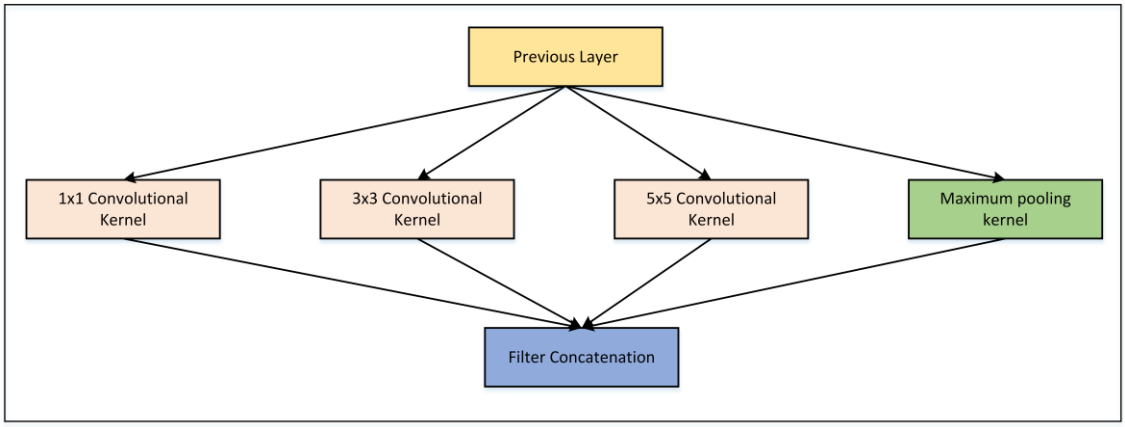

Figure 38 : Naïve Inception module

This module shall enable the processing of visual data at various scales where the large kernels shall capture the features distributed globally, meanwhile the small kernels shall capture the features distributed locally so that abstract features from different scales are aggregated together to next layer

This naïve inception module even if it can cover the optimal sparse structure, the existence of the maximum pooling layer accompanied also by the overall network depth would lead to an exponential growth in the number of learnable weights blowing up the computational resources, thus 1x1 convolutional kernels were applied as a dimension reduction modules for any expensive operation. For instance, before 3x3 kernel, before 5x5 kernel and after the maximum pooling to reduce the number of generated channels within the feature map. Thus depth of network is allowed to be increased without a significant computational penalty. Also, these 1x1 convolutional kernels is associated with applying nonlinear function activation enabling them to have a dual propose. Figure 39 shows the inception with dimension reduction

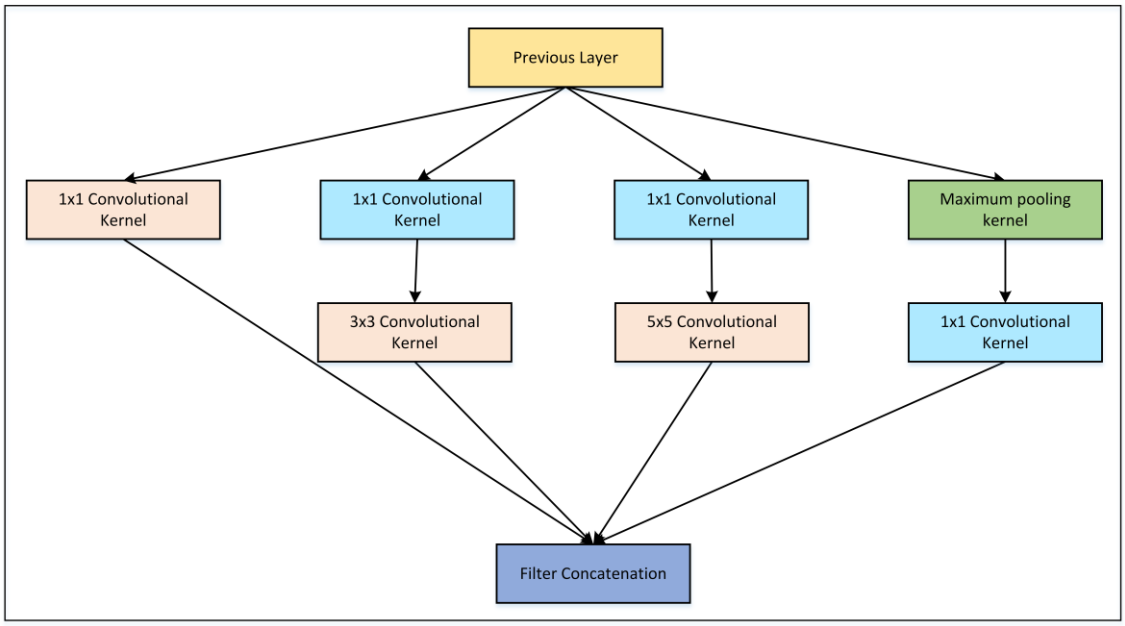

Figure 39 : Inception module with dimension reduction

In general, it is composed of 22 layers divided into three traditional convolutional layers, 18 inception modules and one fully connected layer. It was able to achieve top-5 error of 6.7% on the ImageNet data set with a total 7 million weight parameters.

3.7.2. Second version

The second version was introduced in 2015. It mainly introduces the batch normalization and a new variant from the Inception module.

Batch normalization is the most popular normalization layer used nowadays. The need for normalization arises as the network goes deeper the training is usually complicated given the internal covariate shift fact where the distribution of network parameters changes from one layer to another due to the variation of network activations per layer requiring the layer to adapt continuously to the new distribution. Meanwhile, the training converges faster in case of whitened inputs where inputs have zero mean and unit variance.

Batch normalization seeks to reduce the internal covariate shift by observing the activation output from each layer to whitened it before going to next layer. It can be applied on both fully connected and convolutional layers with a special attention to the convolutional property. It is required that the normalization obeys this property such that different elements at different locations within the feature map are normalized in a similar manner. This shall require the joint normalization of all activation in a mini batch across all location.

However, one drawback back of normalizing each the inputs of layer is it may change what a layer can represent, thus batch normalization introduces two learnable parameters the scale and shift values to ensure the transformation inserted in the network can represent back the identity transform. These parameters are trained with the learnable weights parameters and allow the network to restore back its representation power.

The new inception variant basically decomposes each 5x5 convolutional kernel by a stack of two consecutive 3x3 kernels similar to the VGG network to reduce the number of weights and the associated required computations. In addition to, employing average pooling in some inception modules while in other maximum one is used.

In general, this variant is composed of 32 layers divided into two traditional convolutional layers, 30 inception modules and no fully connected layer. It was able to achieve top-5 error of 7.8% on the ImageNet data set with a total 8.75 million weight parameters.

3.7.3. Third version

The third version was introduced in 2015 and it introduced a new Inception variant which was the first runner up of image classification track within ImageNet challenge.

The new variant scale up the depth of the network while maintaining the computations efficiency through factorizing the large spatial kernels into smaller ones. Convolutions done through large filters as 5x5 filters can span a wide geometric area of the feature achieving more expressiveness in the extracted feature due to its ability to extract more dependences between the generated activations. However, they require more computations when compared to smaller kernels. For instance, 5x5 kernel has 25 parameters while 3x3 kernel has 9 parameters meaning that a 5x5 kernel is 25/9 = 2.78 times computationally expensive than 3x3 kernel. In a vision task, it is expected that adjacent activation units shall generate highly correlated outputs. Thus, these activation

units can be dimensionally reduced followed by spatial aggregation them without losing much information and hence results in similar expressive local representations. Therefore, a 5x5 kernel can be replaced with two sequential 3x3 kernels with a negligible loss in the futures expressiveness.

The 3x3 kernel can even be factorized using the asymmetric kernels (i.e. nx1) to achieve more reduction in the computations. For example, a 3x3 kernel can be decomposed into a 3x1 kernel followed by 1x3 one as shown in Figure 40 with around 33% computation savings.

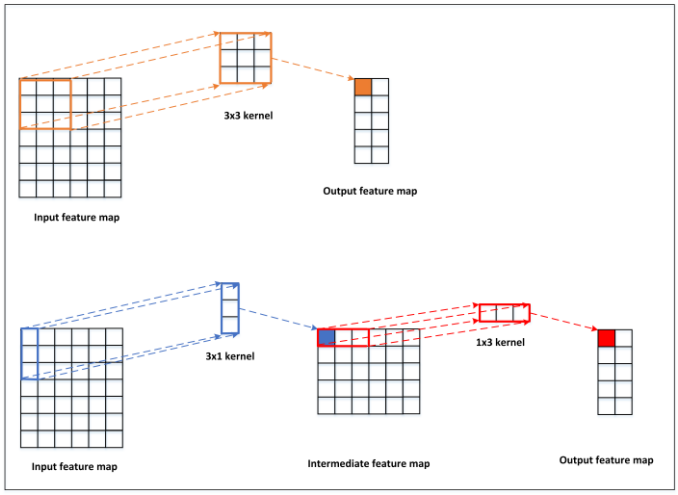

Figure 40 : Decomposing 3x3 kernel into asymmetric kernels

Thus, hypothetically any n x n kernel can be replaced by 1xn kernel followed by nx1 kernel and this led to the introduction of a new inception module shown in Figure 41. However, practically this type of factorization doesn't perform well at the network early layers requiring their usage at the mid to the end layers.

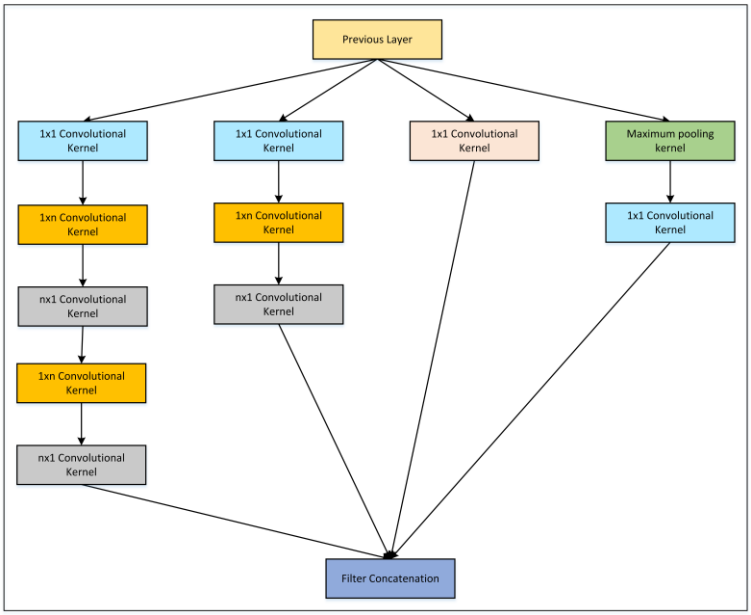

Figure 41 : New Inception module with nx1 and 1xn factorized kernels

In general, this variant is composed of 42 layers that was able to achieve top-5 error of 5.7% on the ImageNet data set with a total 29.3 million weight parameters.

3.7.4. Fourth version

The fourth version was introduced in 2016 and it introduce a new variant that combines the Inception module with the residual connections introduced in [29] as shown in Figure 42.

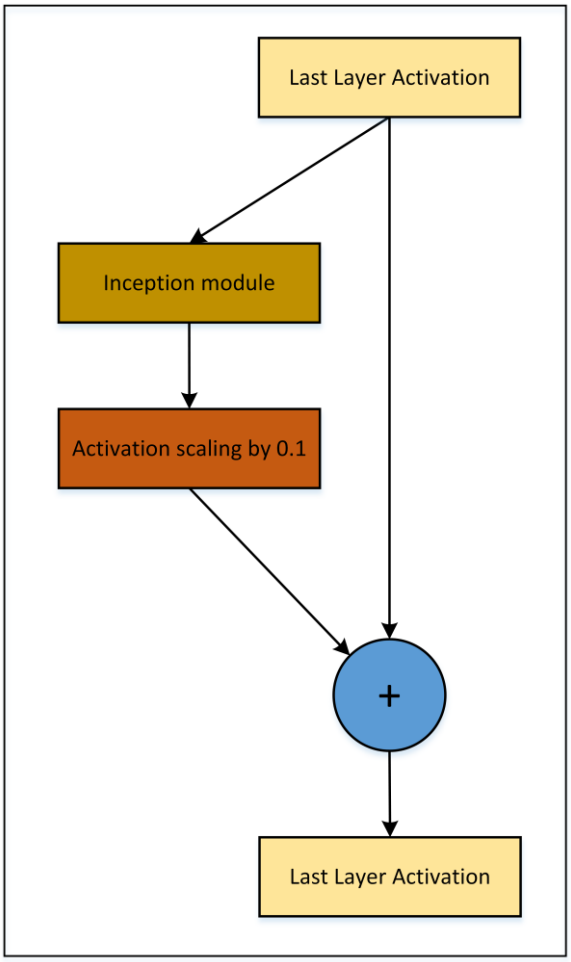

Figure 42 : Inception module accompanied by residual connection

In general, this variant is composed of 164 layers that was able to achieve top-5 error of 4.9% on the ImageNet data set with a total 55.93 million weight parameters.

3.8. ResNet

Also known as Residual Net [29] was introduced in 2015 and was able to win all the tracks within the ImageNet challenge. It is considered the first network to exceed the human level performance in the ImageNet challenge with top-5 error below 5%. Since its introduction, it was followed by another version [102].

3.8.1. First version

The first version introduces the shortcut module and similar to the previous networks it attempts to increase the depth of the network.

Depth aspect is proven to be crucial for network performance, however straight forward stacking of more layers had been shown to degrade the performance once the network starts to converge where the accuracy starts to saturate followed by a rapid degradation. Such degradation is not argued to overfitting only but also the optimizer may have faced difficulties during resolving the cost function. This can be explained through the assumption of having a shallow architecture where a deeper counterpart architecture that adds several layer onto it can produce no higher training error compared to the shallow exits if the added layers are identity mapping ones.

The shortcut module is inspired from the aforementioned degradation problem where the optimizer may have faced difficulties when trying to approximate the multiple nonlinear transformation layers into identity mappings. As shown in Figure 43, it contains an identity connection to allow the network to skip the convolutional layers such that if the optimal function to be learnt is closer to the identity mapping, the optimizer shall easily find the perturbations with reference to an identity mapping rather than to learn the function. Furthermore, this module doesn't add any extra parameters.

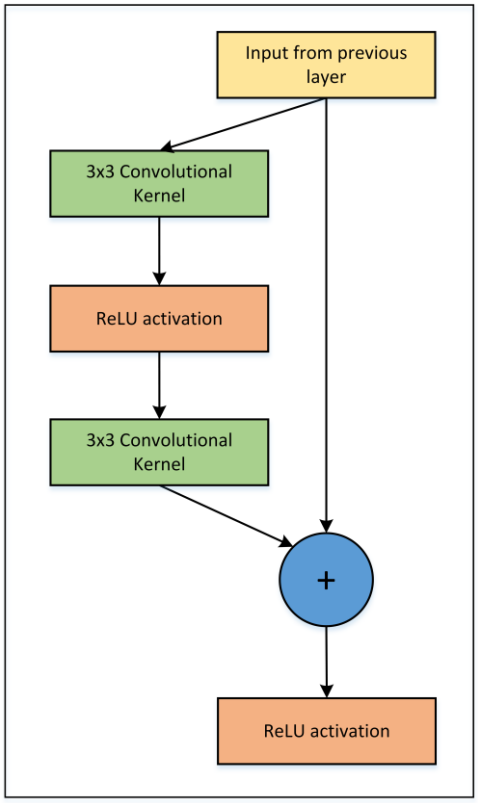

Figure 43 : Shortcut module

Another module is introduced which is the bottleneck module. It modifies the shortcut module to reduce the learnable weight parameters as well as the training time through the usage of 1x1 kernel. This is done as shown in Figure 44 by replacing the two layers stack with a three one in which the three layers are stacked as 1x1 kernel, 3x3 kernel and 1x1 kernels where the 1x1 kernels are used to reduce and then restore the

dimensions, leaving the 3x3 kernel as a bottleneck with smaller input and output dimensions.

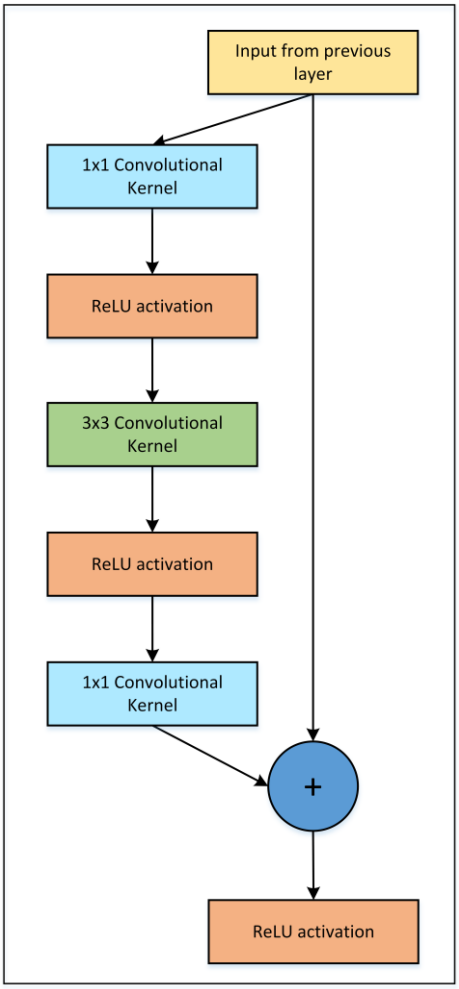

Figure 44 : Bottleneck module

In general, ResNet follows the same philosophy of the VGG where it is constrained to use only 3x3 kernel, all layers with the same output feature map dimension shall have the same number of filters and the number of filters is doubled as the network goes deeper with the feature map size is halved. The main modification is the insertion of the bottleneck modules which convert the network to a residual version. It has three popular variants ResNet-50, ResNet-101 and ResNet-152.

ResNet-50 is composed of one convolutional layer, 16 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 25.5 million weight parameter and top-5 error of 5.25% on the ImageNet data set.

ResNet-101 is composed of one convolutional layer, 33 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 44.5 million weight parameter and top-5 error of 4.6% on the ImageNet data set.

ResNet-152 is composed of one convolutional layers, 50 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 60 million weight parameter and top-5 error of 4.49% on the ImageNet data set.

3.8.2. Second version

The second version was introduced in 2016. It mainly analyzes and conducts some experiments on the residual network attempting to create a direct path for the propagation of information within the entire network instead of the shortcut module only. It also introduces a new variant from the ResNet.

A new shortcut module as well as its counterpart bottleneck module are introduced which are shown in Figure 45 (a) and (b) respectively where the identity connections are kept as the direct path for information propagation, meanwhile the nonlinear activation function are rearranged such that the ReLU and the added batch normalization are used as a pre-activation functions such that the activation is moved to residual mapping pathway.

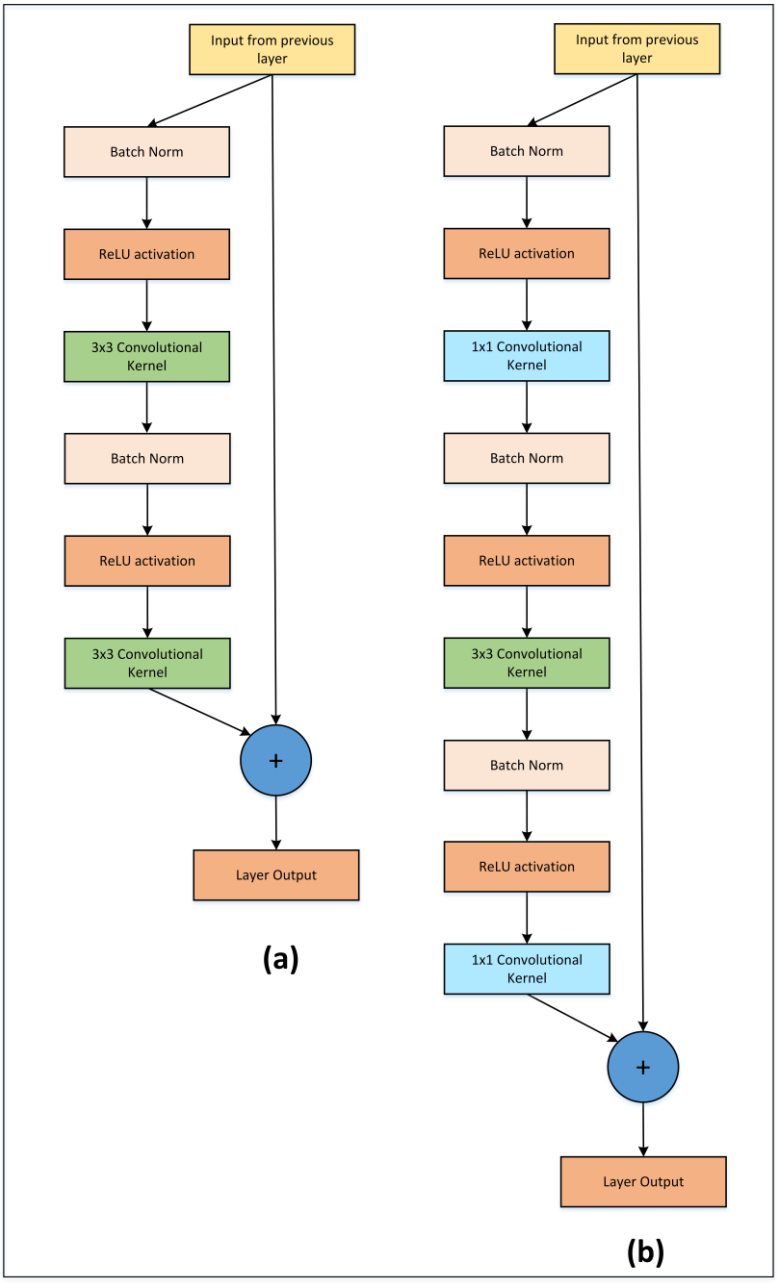

Figure 45 : (a) Modified shortcut module (b) Modified bottleneck module

In general, it has introduced two new variants ResNet-152 and ResNet-200.

ResNet-152 is composed of one convolutional layers, 50 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 60 million weight parameter and top-5 error of 5.5% on the ImageNet data set.

ResNet-200 is composed of one convolutional layers, 66 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 64.7 million weight parameter and top-5 error of 4.8% on the ImageNet data set.

3.9. Conclusion

Figure 46 summarizes the evolving networks since AlexNet where obvious trends across these networks can be observed

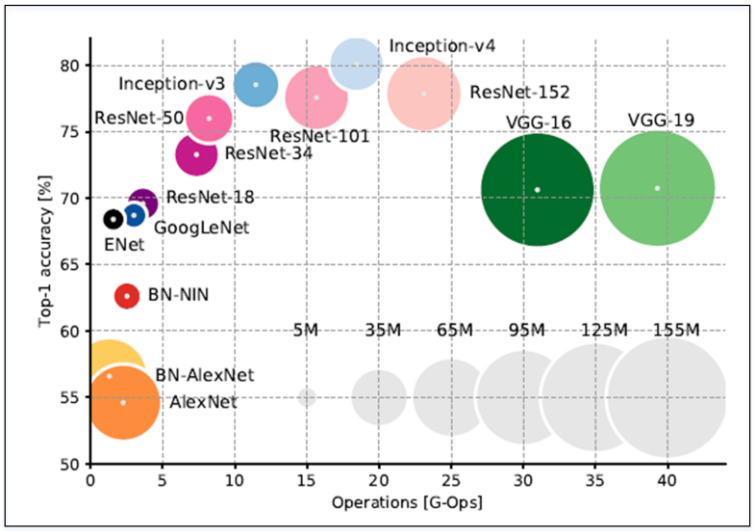

Figure 46 : Different networks compared according to their size, number of operations and Top-1 accuracy from [106]

Firstly, the attempt to improve the network accuracy through increasing the size of the networks in terms of depth which is reflected in number of layers within the network as well as the width which is reflected in number of units per layer. The network size increase can be beneficial through the increase of the number of nonlinear functions applied allowing the network to be more discriminative and increasing the number of abstracted learned representation hierarchy. However, this shall come with a price in terms of dramatically increase in the required computational resources to train the network and the network tendency to over fit. To overcome these problems while being able to increase the network size, computation efficient networks which start to modify shape of layers and their connection were innovated as shown in the Inception and ResNet networks

Secondly, the number of fully connected layers are reduced moving most of the computations and learnable weights to the convolutional layers. Moreover, networks like Inception doesn't include a one

Thirdly, the network kernel size tends to be more compact. A kernel size can vary from very large size (i.e. 11x11 as in AlexNet) to a very small one (i.e. 1xn or nx1 as in Inception). Decomposing large kernels into a set of cascaded smaller ones can reduce the

computation complexity and the number of learnable parameters through the replacement of the loose and over parametric kernels with compact ones, meanwhile applying these smaller kernels sequentially can maintain the overall effective receptive field achieving almost the same network performance. Moreover, kernels decomposition can be beneficial in increasing the number of nonlinear transformation applied enhancing the network capability to be more discriminative

Chapter 4 : Exploring Convolutional Neural Networks Different Layers

As illustrated in the previous chapter, the current trend in designing CNN is to modify the structure of its layer either by introducing a new convolutional kernel (i.e. 1x1 convolution) or the connection between different kernels within its layer (i.e. Inception [98] or ResNet[29]) before starting to increase the depth of the network. Network modification on the architecture level had shown to be fruitful achieving a significant enhancement in the performance

In this chapter, exploring different modification in the CNN layers whether the convolutional or the pooling was done with the aim to introduce a new mapping function.

4.1. Basic Setup

To start exploration, it is required to choose the application task, followed by defining the target data set, then switching to choose an efficient framework and finally defining the platform to run out the experiments upon it.

The selected application shall be image classification given it is the common task used to evaluate the evolving networks as well as being the basis for other computer vision tasks such as object detection and localization.

The picked out Data set shall be CIFAR-10[63] where it is considered as an acceptable data set used in experimenting some of the state of art networks as NiN[105] and ResNet[29], meanwhile having an average complexity when compared to MNIST[62] which is very easy and tiny one and the ImageNet[64] which is a huge data set that requires a very expensive computational infrastructure

The chosen Framework shall be Keras with TensorFlow as backend [107]. Keras is a widely adopted framework with a lot of online supports. The keras flow as well as its associated key image data generator class is described in Appendix one

Among the different available computing platforms available, google compute engine and amazon web services were used interchangeably across this work. These platforms are equipped with a high end GPUs such as V100 enabling deep networks experimenting and training. More about the computing platforms can be found in Appendix two

4.2. Baseline network

The network used in the experiments of this chapter is inspired from VGG [97], where it follows its footsteps as shown in Figure 47 from fixing the kernel size to 3x3 to using two convolutional layers with same number of kernels before halving the feature map through the usage of a maximum pooling layer. A fully connected layer is then applied at the end to generate the logits of the target classified class

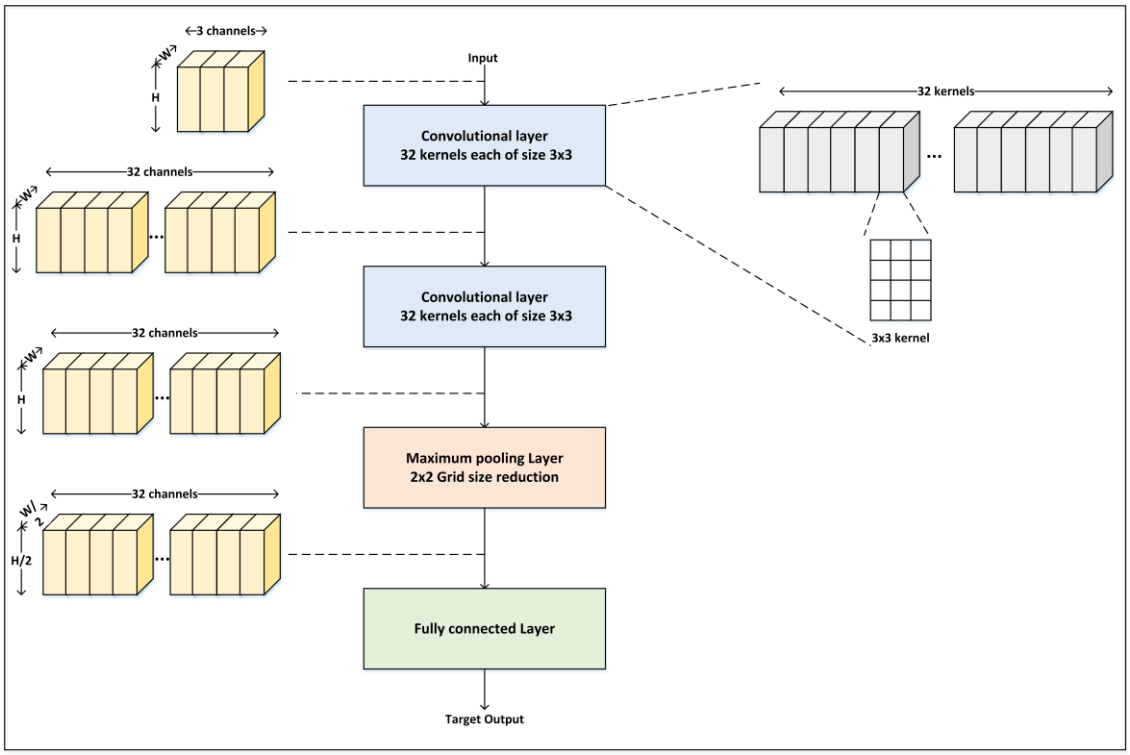

Figure 47 : Baseline network

4.3. Convolutional layer modification

4.3.1. Pseudo Rotated Kernels

Reviewing back the basic convolution operations as shown in Figure 48 where spatial image filtering is done through convolving a trainable weight kernel with an input image to generate the feature map going to next layer.

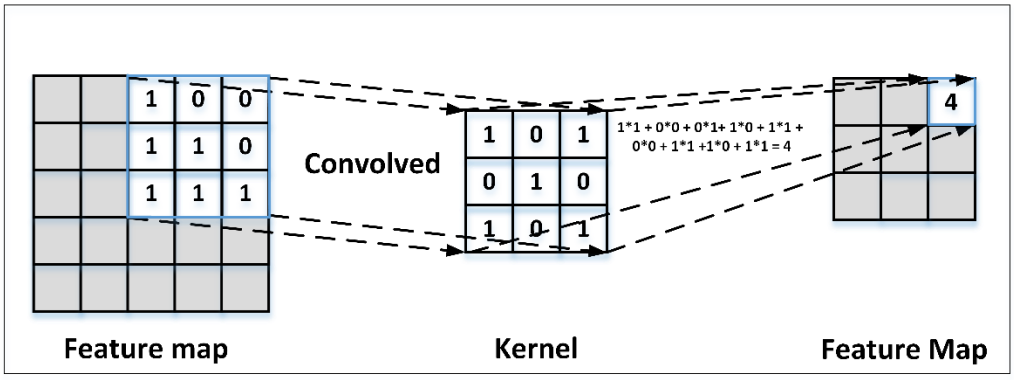

Figure 48 : Basic convolutional operation

From an operation point of view, this can be shown as modifying the intensity of a pixel according to the intensities of the neighboring pixels. Another point of view, is the

mathematical one, where this operation is actually a cross correlation one where actual convolution requires rotating the filter by 180 degrees before convolving it with the input.

This is the initial inspiration of applying the pseudo rotation kernels, where the network can benefit from the usual cross correlation function in addition to allowing it to perform an actual convolutional one using a rotated kernel with 180 degrees enhancing its capabilities in extracting more useful features. Figure 49 shows a cross correlation kernel with a zero degree rotation and its convolution kernel pair with 180 degree rotation).

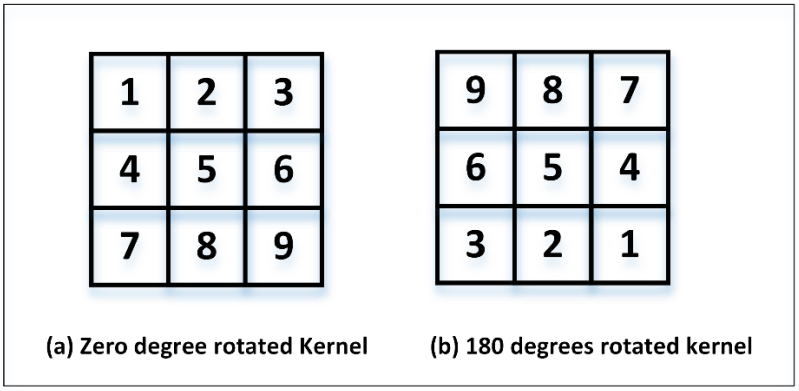

Figure 49 : (a) Zero degree rotated kernel (b) 180 degree rotated kernel

Using this pair of kernels, the baseline network can be modified as shown in Figure 50 with a note here is the generated feature map is almost doubled.

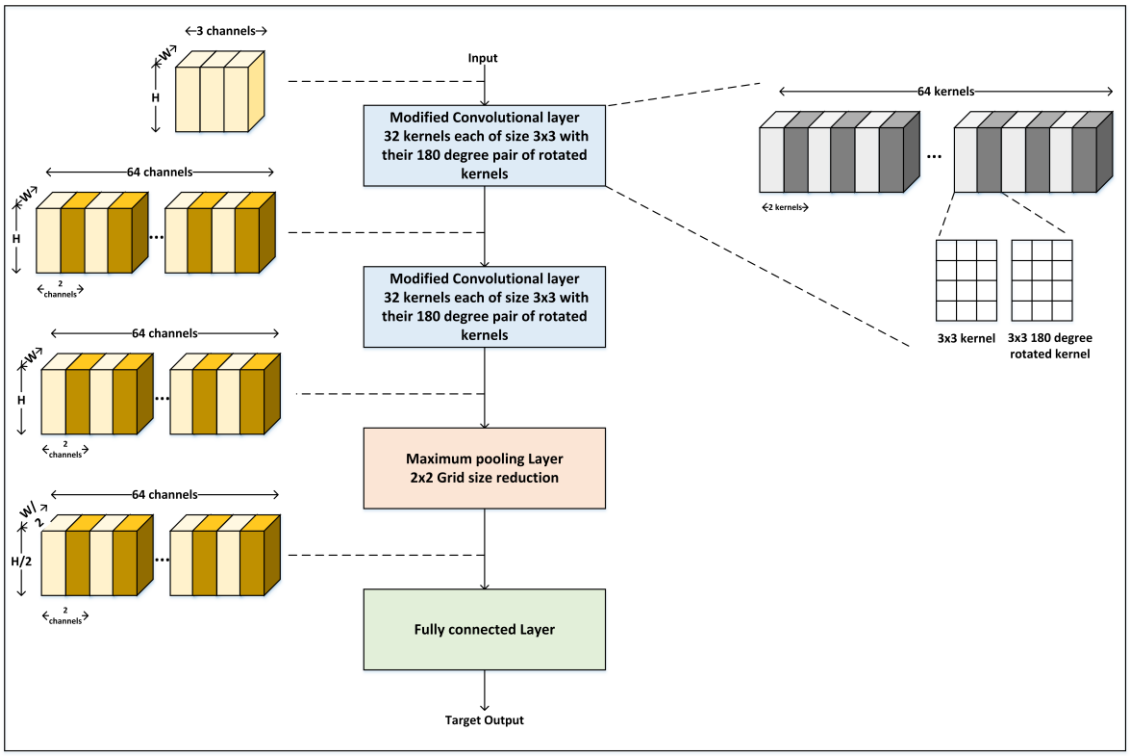

Figure 50 : Modification to baseline network to account for the 180 degree rotated kernel

Moreover, the idea of rotating kernels can get more insights from image processing techniques, where the feature detectors like Robert Cross edge detection or Sobel edge detectors are based on rotating kernels where a pair of zero degree kernel and a 90 degree rotated one are applied to extract the features. Hence, generalizing the rotating kernels to have a 90 degree rotated kernels as shown in Figure 51 would be fruitful.

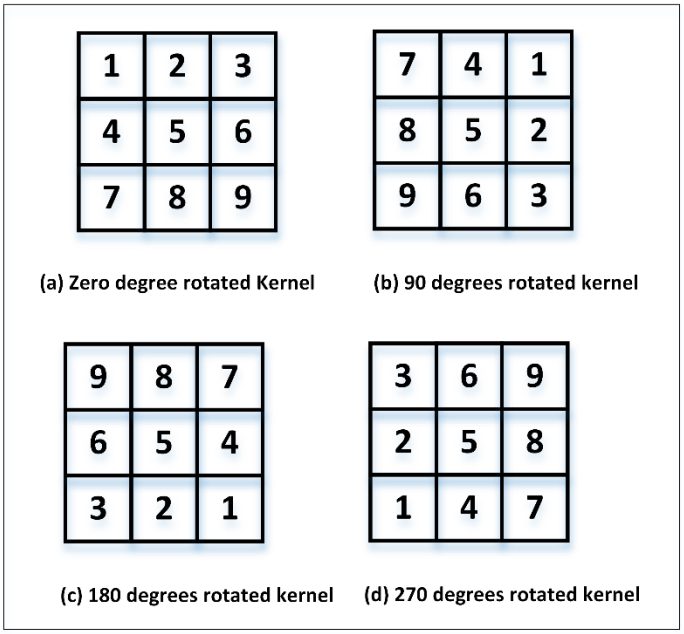

Figure 51 : Pairs of 90 degree rotated kernels starting from(a) a zero one to (d) 270 degree rotated kernel

Another modification for the baseline is required in accordance to applying the 90 degree rotated kernels one as shown in Figure 52.

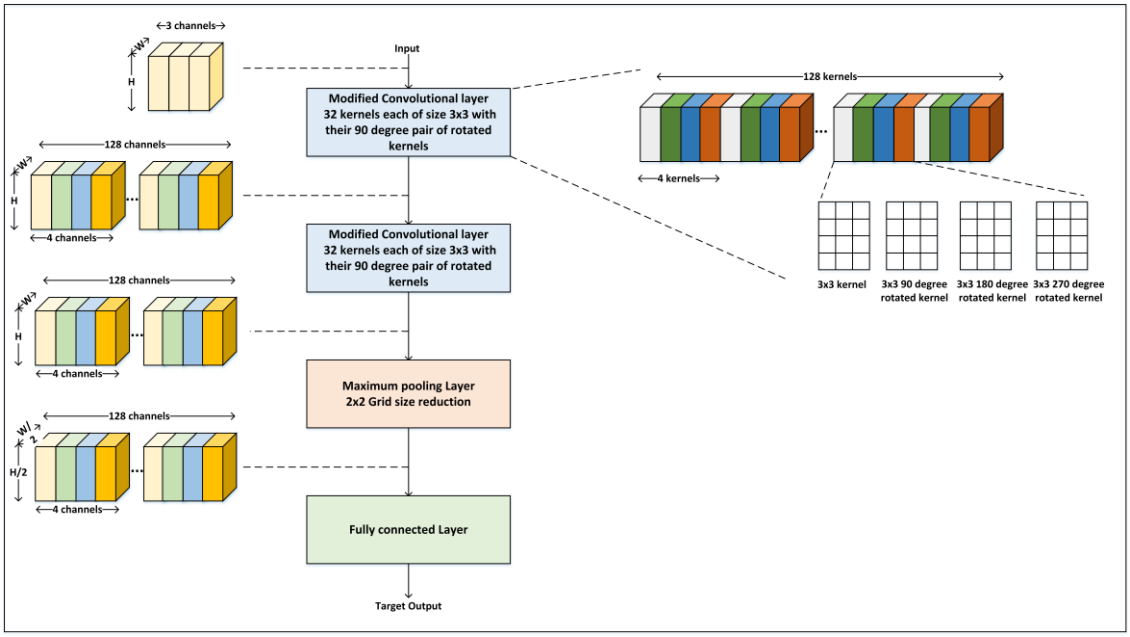

Figure 52 : Modification to baseline network to account for the 90 degree rotated kernel

However, to squeeze the idea more, an attempt to rotate the zero degree kernels by 45 degree won't be feasible as it requires the kernel shape to be a trapezoid, thus to overcome this limitation an approximation can be done to divide the 90 degree rotation into two steps as shown in Figure 53 where one step is to rotate the kernel in the required 90-degree manner but with a shuffled kernel and the second step is to rearrange the kernel to obtain the 90 degrees rotation kernel. The first step can be considered as a pseudo 45 degree rotation meanwhile the second step is usual 90 degree rotation.

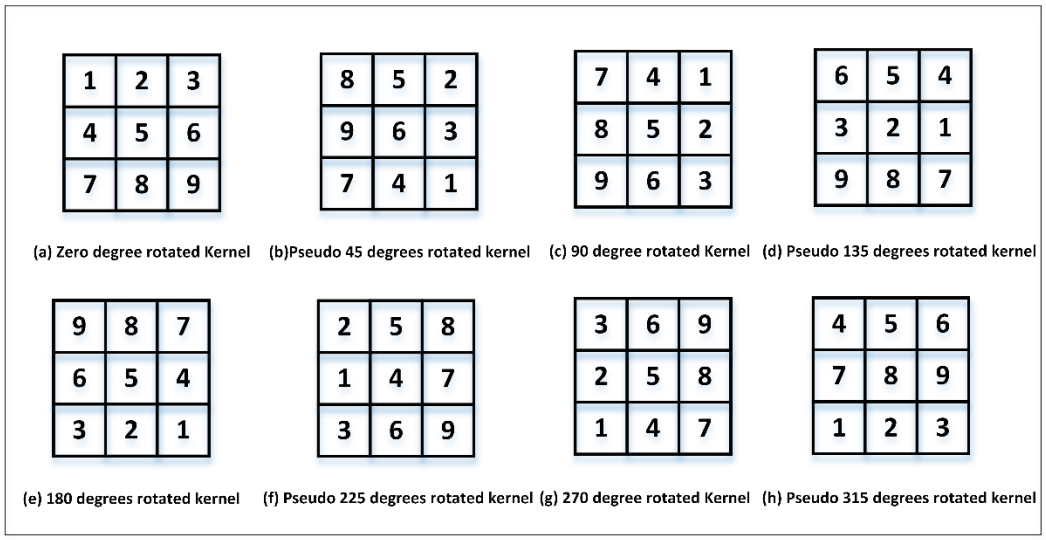

Figure 53 : Pairs of pseudo rotated 45 degree kernels starting from(a) a zero one to (h) 315 degree rotated kernel

The modification of the baseline to account for the pseudo 45 degree rotation can be shown in Figure 54.

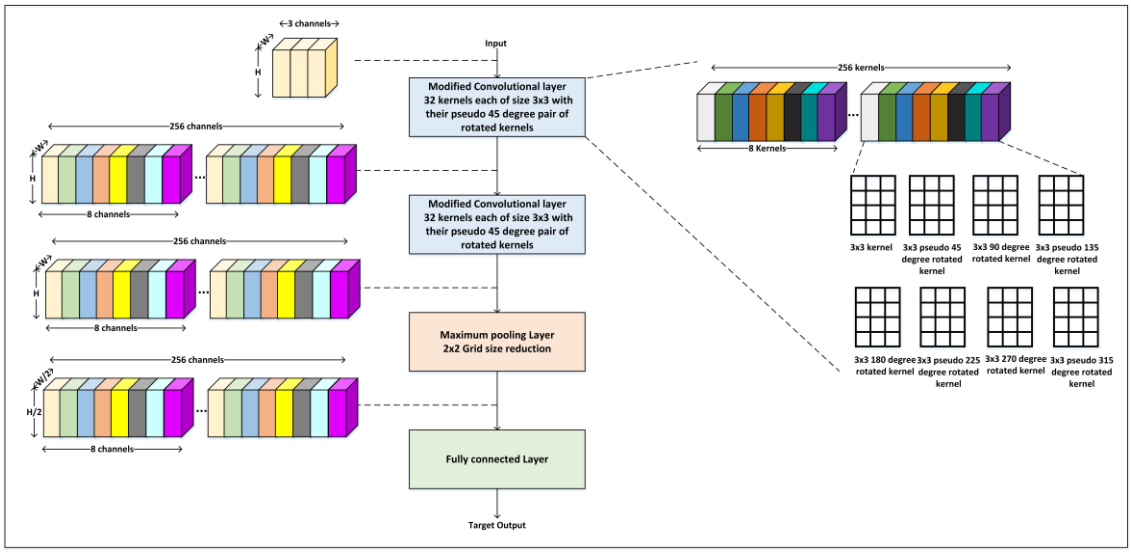

Figure 54 : Modification to baseline network to account for the 45 degree rotated kernel

When analyzing the aforementioned approximation, it can be shown that the step of rotating the kernel into a certain direction is a mandatory one given that the 90 degree multiples have a straight forward structure while the pseudo intermediate rotation step which includes kernel shuffling is an arbitrary one given the way of arranging the kernel was a subjective one.

This would lead to generalizing the pseudo rotation steps more by assuming their shuffling before reaching the 90 degree multiples can be divided into more fine steps that can cover all the available shuffles. This is what pseudo 15 degree rotation does as shown in Figure 55.

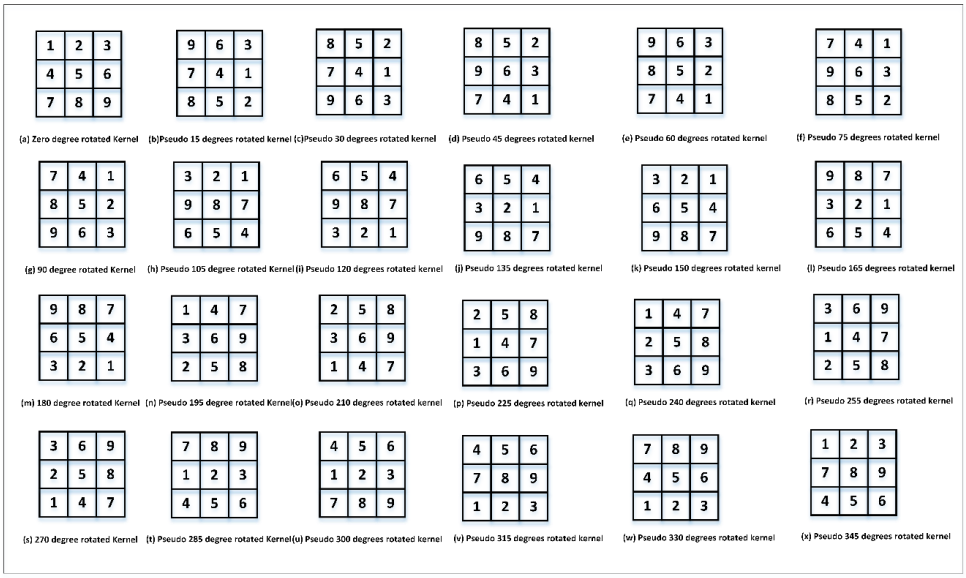

Figure 55 : Pairs of pseudo rotated 15 degree kernels starting from(a) a zero one to (x) 345 degree rotated kernel

Also, the modification of the baseline to include the pseudo 15 degree rotation can be shown in Figure 56.

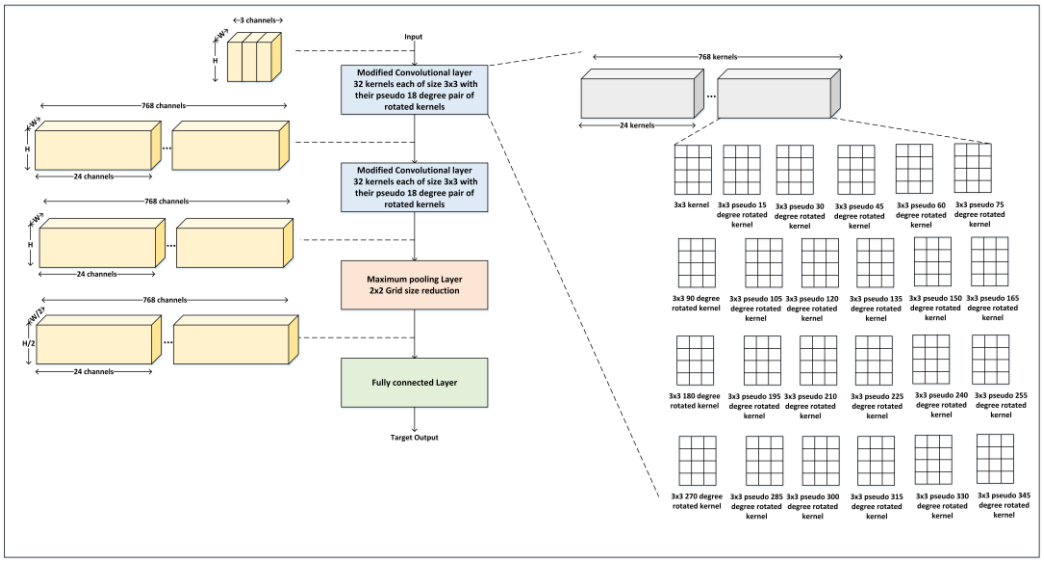

Figure 56 : Modification to baseline network to account for the 15 degree rotated kernel

The 15 degree pseudo rotated kernels actually would lead to the creation of the pseudo rotated kernels design space as shown in Figure 57, where there is a pool of pseudo rotation kernels along the rotation circle with an arbitrary choice during the design of the network to choose which of them to be applied. This can be viewed as adding a new kernel type in the network optimization problem similar to how NiN[105] added a new dimension in optimizing the kernels through the introduction of the 1x1 kernel.

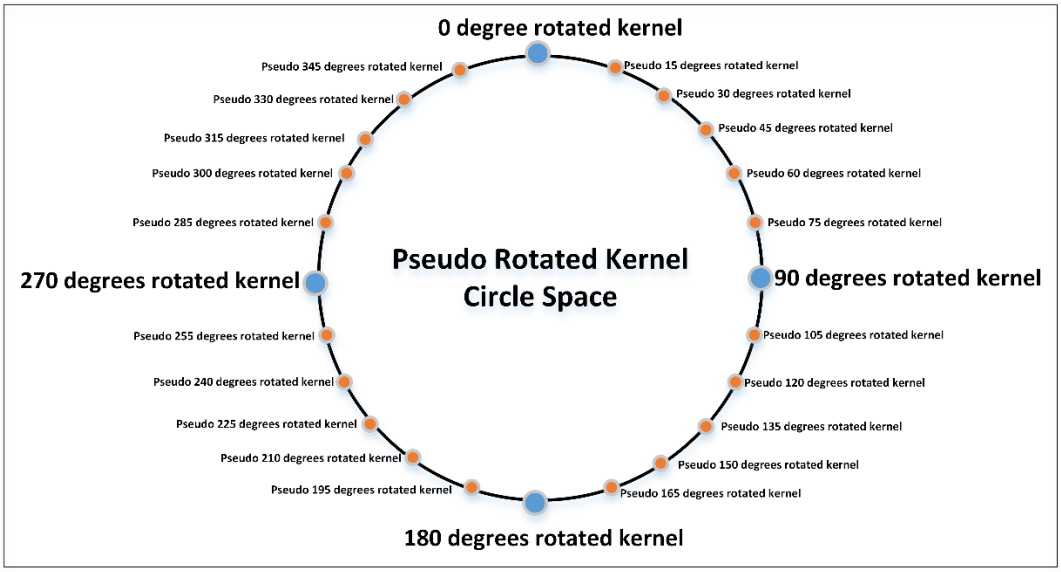

Figure 57 : Pseudo rotated kernels circle design space

Table 1 shows the comparison of achieved accuracy from the baseline network and its modified versions as well as the number of increased parameters after training them for 10 epochs

Network	Top-1 error	Number of parameters/ Computation ratio
Baseline	34.06 %	1x
Modified with 180 degree kernels	31.56 %	2x
Modified with 90 degree kernels	33 %	4x
Modified with 45 degree kernels	33.4 %	8x
Modified with 15 degree kernels	33.7%	24x

Table 1 : Comparison between Baseline network and its pseudo rotated modified versions

The results obtained from the modified networks are promising given that they showed some enhancements in the accuracy with only few number of epochs

The intuition here is that the pseudo rotated kernels interact with the affine transformation which is the core foundation of the ML theory through its translation and rotation methods. Translation method is established in the CNN by means of the convolutional kernels (recall it is actually a cross correlation one) where the pseudo rotated filters widen this method by enabling the network to perform the usual cross correlation function accompanied by the actual convolution one by means of the 180 degree rotated kernel, meanwhile the pseudo rotating kernels enhance the rotation method by providing a set of arbitrary chosen rotated kernels at each layer. Moreover, the pseudo rotated kernels increase the robustness of translation invariance property of the network by providing the feature map rotated in several ways as if the network is capable to rotate the feature map at each layer. This may be viewed as if the network has become self augmented where it has its own self augmentation methods.

A final note to be mentioned is that modifying the kernel size to larger sizes (i.e. 5x5 kernels) to have more pseudo rotation degree steps is assumed to be non-beneficial given the current shown benefit from the state of the art networks in making the kernel size more compact. Furthermore, increasing the kernel size will come with a huge computational cost penalty from the larger kernel its self and its associated pseudo rotated ones making the training process non feasible. For instance, a 5x5 kernel with its 90 rotated kernel shall require 100 learnable weigh while the 3x3 kernel shall require 36 only meaning that the a single 5x5 kernel requires approximately 2.7 extra computation power to be trained.

4.3.2. Kernels Mathematical derivations

Another approach to modify the convolutional layer is to introduce some mathematical relations between different kernels in analogous to how the MFFCs filter bank in speech recognition is constructed to extract the features. This filter bank performance was enhanced through correlating different filters together by means of averaging each two successive one to generate a new one that can benefit from the previous and the subsequent filter. This can be viewed as introducing an intermediate kernel that can hopefully generate a new useful feature from the already feature trained kernels.

One potential relation as shown in Figure 58 is to generate a new kernel between every two successive kernels through either averaging them or using one of the basic operations such as addition, subtraction, multiplication and division or a complex one such as geometric mean, root mean square and the logarithmic mean.

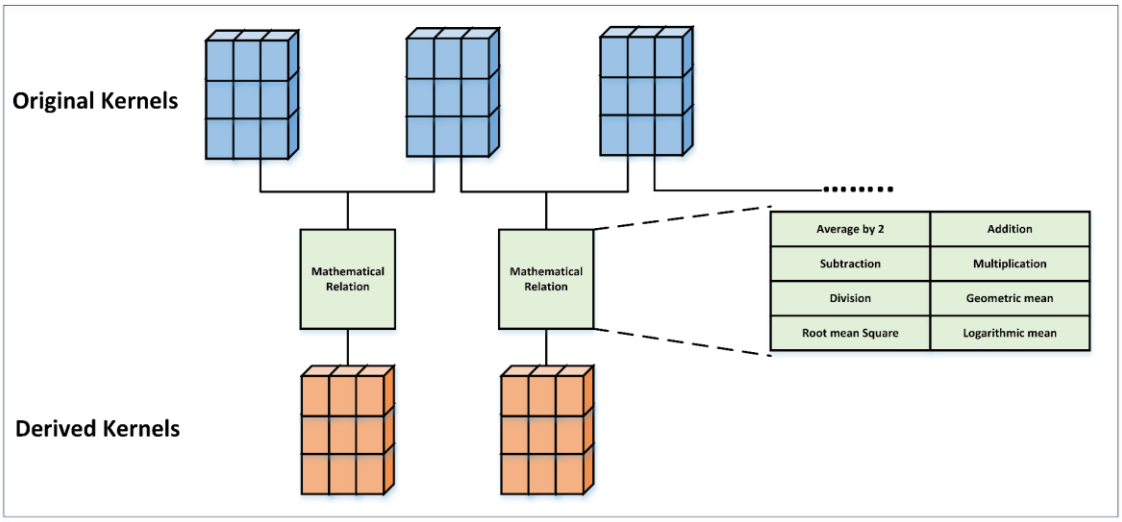

Figure 58 : Adding a derived kernel between every two successive kernels

The intuition here is that if the kernels are concentrating on correlated regions to generate different features then introducing an intermediate one can benefit from both of them to capture a new feature that would be captured only using a larger kernel or when processed in the next layer. This can be beneficial to the learning process of the network as if it is equipped with a larger kernel. The modification to the base line network can be shown in Figure 59.

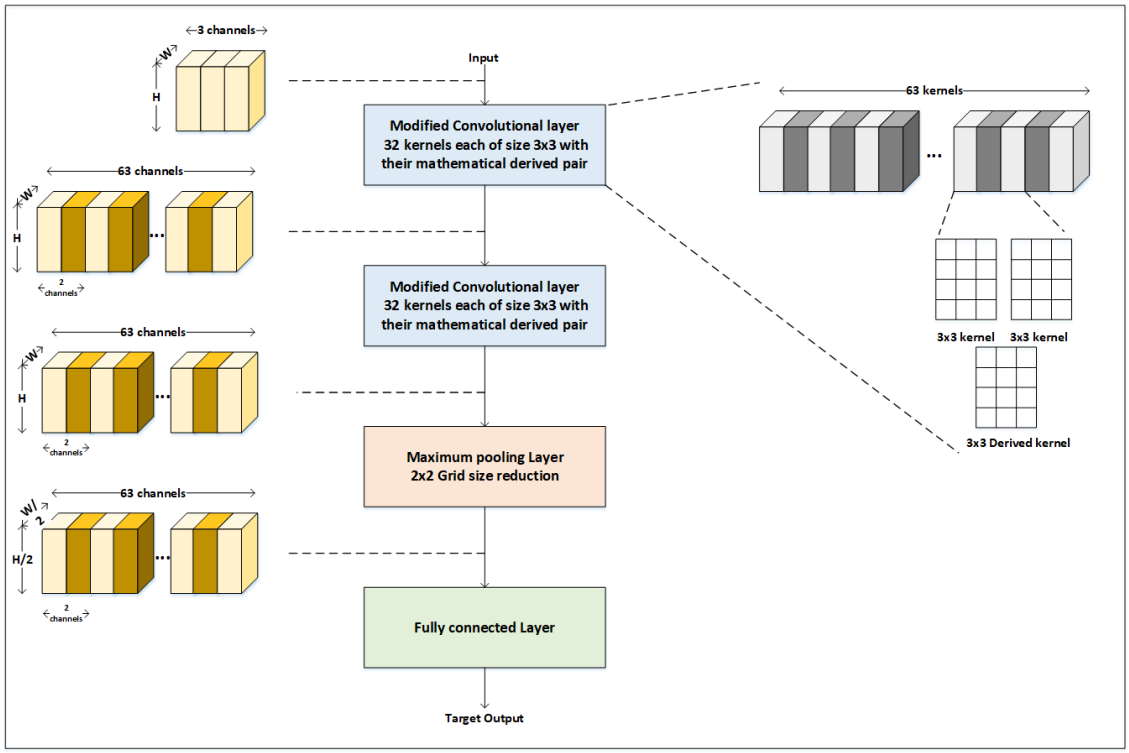

Figure 59 : Modification to baseline network to account for the kernels derived from every two successive ones

Table 2 shows the comparison of achieved accuracy from the baseline network and its modified versions as well as the number of increased parameters after training them for 10 epochs

Network	Top-1 error	Number of parameters/ Computation ratio
Baseline	34.06 %	1x
Modified with Average by 2	34.4 %	1.97x
Modified with Addition	35.5 %	1.97x
Modified with Subtraction	34 %	1.97x
Modified with Multiplication	34.2 %	1.97x
Modified with Division	90 %	1.97x
Modified with Geometric mean by 2	39 %	1.97x
Modified with Root mean square by 2	42.25 %	1.97x
Modified with Logarithmic mean by 2	90 %	1.97x

Table 2 : Comparison between Baseline network and its modified versions to account for derived kernels between every two successive kernels

The results obtained are a disappointing one as the network didn't benefit from the introduced kernels showing that the kernels weren't correlated enough to allow the generation of new kernels that can benefit from how the kernels overlap on the same region to generate different features.

One modification to the generated kernel is to increase its size allowing more regions to be overlapped and hence increasing the probability of correlating the kernels together. The kernel size is increased to 5x5 as shown in Figure 60

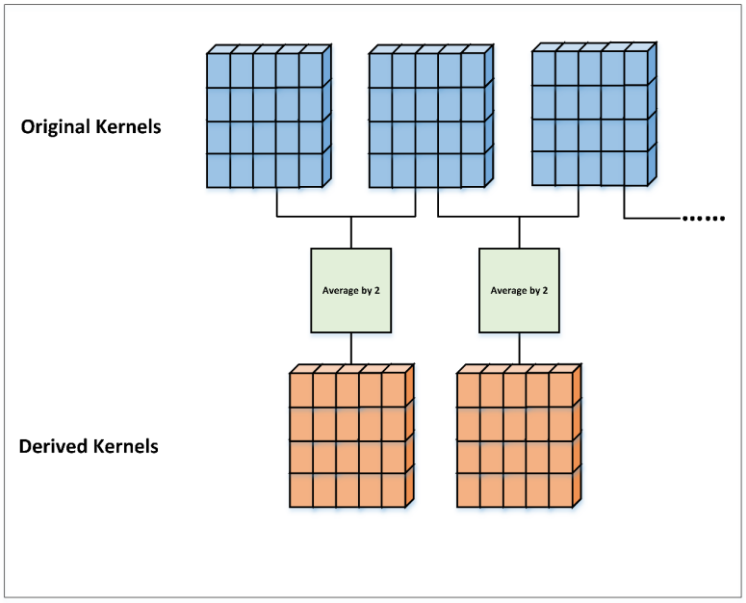

Figure 60 : Adding a derived kernel between every two successive kernels with an increased size to 5x5

Also a modification to the base line network is done in accordance which can be shown in Figure 61.

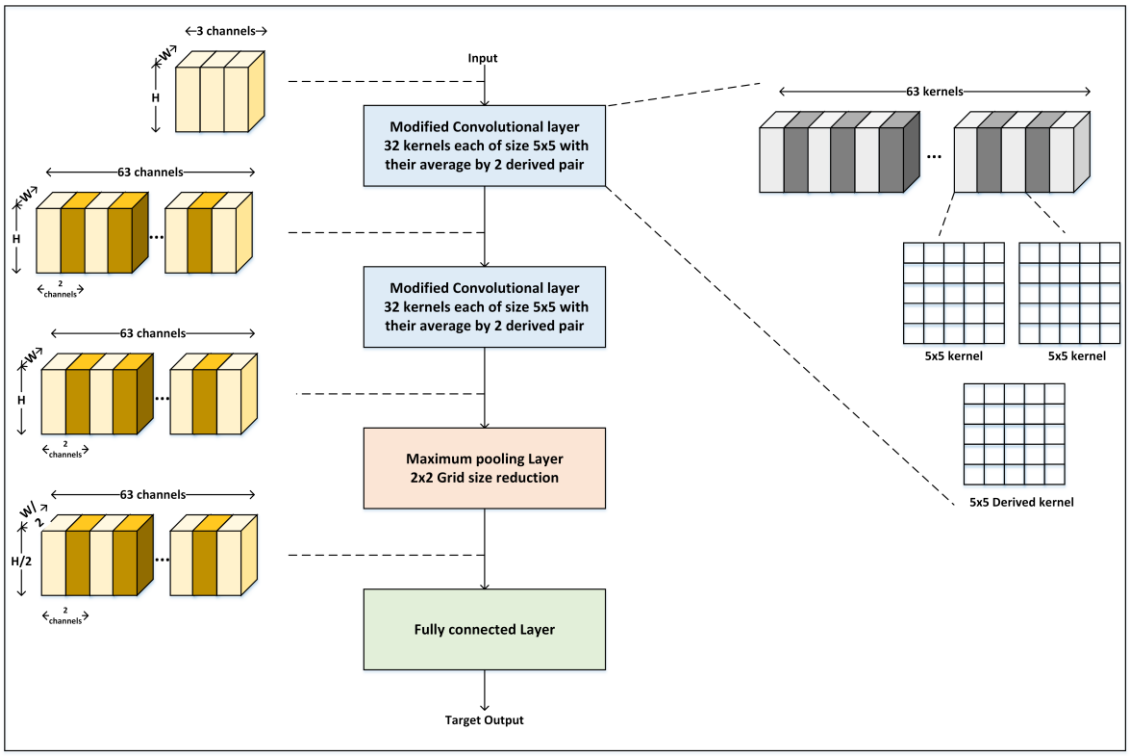

Figure 61 : Modification to baseline network to account for increasing the derived kernels from every two successive kernels size to 5x5

Table 3 shows the comparison of achieved accuracy from the baseline network and its modified versions as well as the number of increased parameters after training them for 10 epochs

Network	Top-1 error	Number of parameters/ Computation ratio
Baseline	34.06 %	1x
Modified with Average by 2	38.3 %	5.46x

Table 3 : Comparison between Baseline network and its modified version to account for increasing the size of the derived kernels between every two successive kernels to 5x5

The results didn't improve showing that this may be the wrong dimension of modification.

However, another dimension is to increase the window of averaging instead of 2 only may be using 4 or 8 or even start to bias the averaging using a weighted one may enhance the accuracy. Figure 62 shows the averaging by 4, while Figure 63 shows the averaging by 8. The modification to the baseline in accordance to averaging by 4 and 8 can be shown in Figure 64 and Figure 65 subsequently.

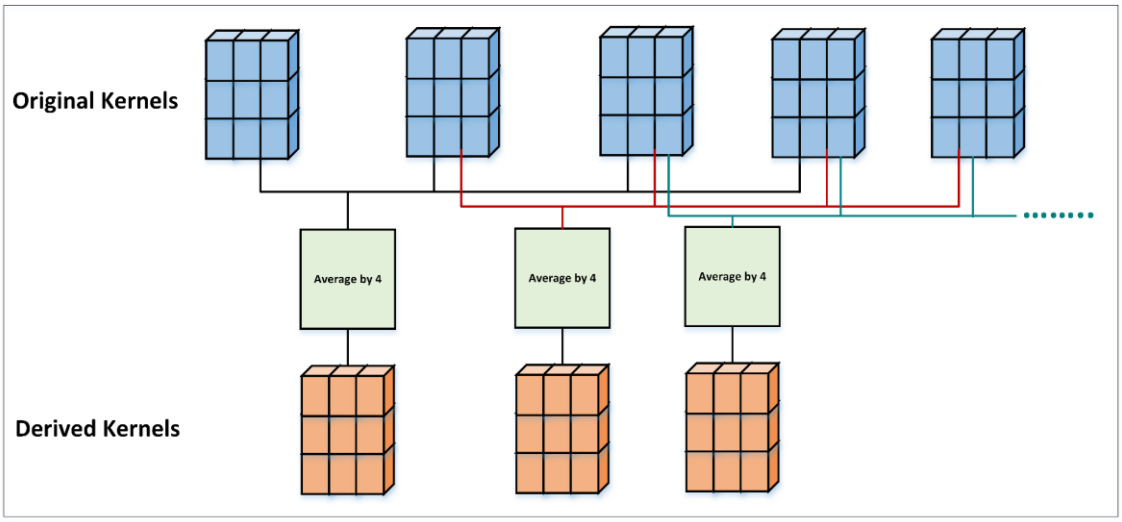

Figure 62 : Adding a derived kernel between every four successive kernels

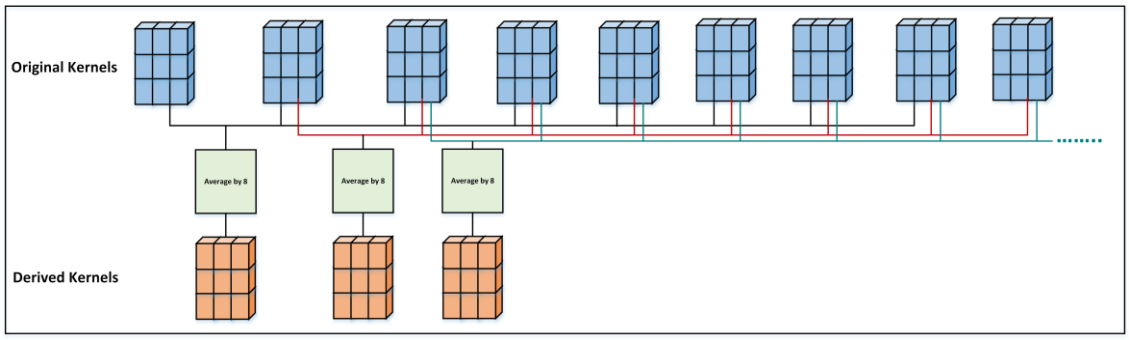

Figure 63 : Adding a derived kernel between every eight successive kernels

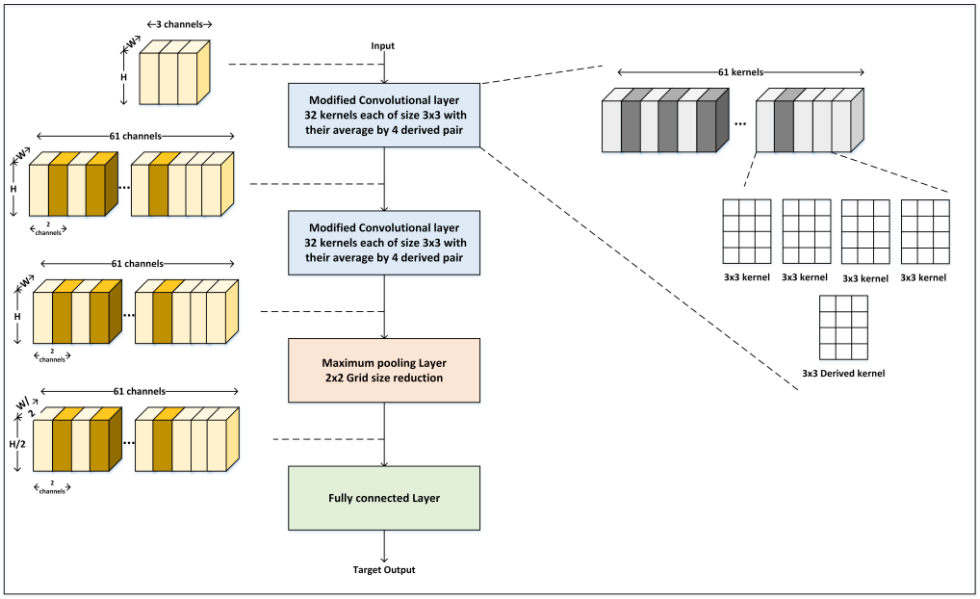

Figure 64 : Modification to baseline network to account for the kernels derived from every four successive ones

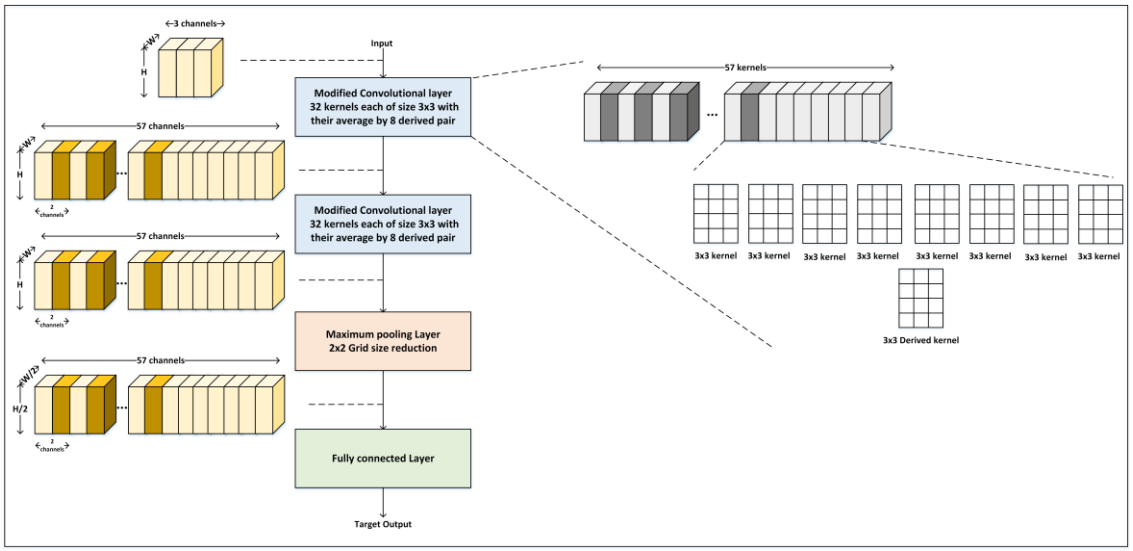

Figure 65 : Modification to baseline network to account for the kernels derived from every eight successive ones

Table 4 shows the comparison of achieved accuracy from the baseline network and its modified versions as well as the number of increased parameters after training them for 10 epochs

Network	Top-1 error	Number of parameters/ Computation ratio
Baseline	34.06 %	1x
Modified with Average by 4	34.1 %	1.9x
Modified with Average by 8	34.03 %	1.78x
Modified with Weight Average by 4	34 %	1.9x
Modified with Weight Average by 8	34.34 %	1.78x

Table 4 : Comparison between Baseline network and its modified version to account for increasing window of derived kernels to be four and eight successive kernels

The results didn't show any improve which suggests that this method mightn't be beneficial for image processing in contrast to speech recognition. This can be regarded to the fundamental difference between both of them, where image processing is spatially correlated while speech recognition is timely correlated. Thus, correlating the kernels can benefit from the sequential nature of the speech recognition and the subsequent kernels can be correlated together, meanwhile image processing has spatial nature where the intensity of a group of pixels are correlated to each other requiring the kernels to be spatially correlated which is inherited in the convolutional kernels through sharing weights

4.4. Pooling layer modification

These layers are used to reduce the feature map dimensions mainly the height and width with the maximum and average layers being the widely used nowadays.

Another similar method which is used in image processing is the Median layer. As shown in Figure 66 it is similar to the average layer where it applies the averaging on the pixel to subsample the feature map to the required dimension. However instead of averaging all the window pixels, it rearranges the window such that it can focus only on the middle ones allowing their average only (i.e. obtaining the mean). The advantage of this layer over the others is in its ability to smooth the feature maps where it discards any intensity overshoot in the pixels within the window that can be viewed as unrepresentative to the surrounding pixels.

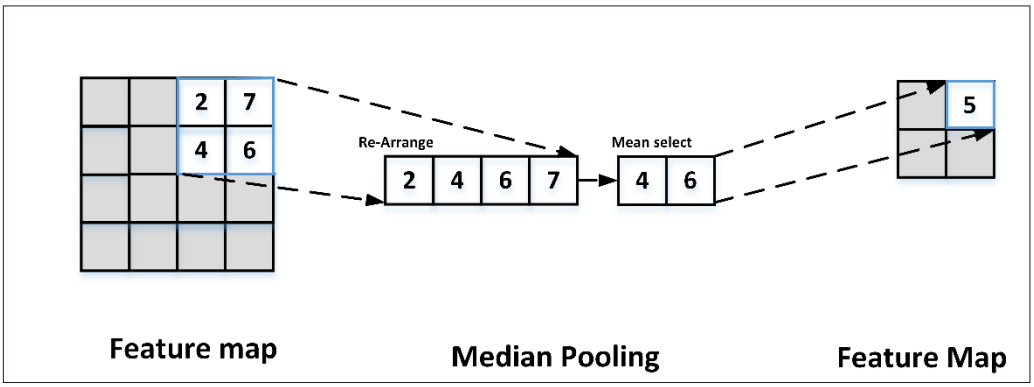

Figure 66 : Median Layer

The modification of the base line network in accordance to using median layer can be shown in Figure 67.

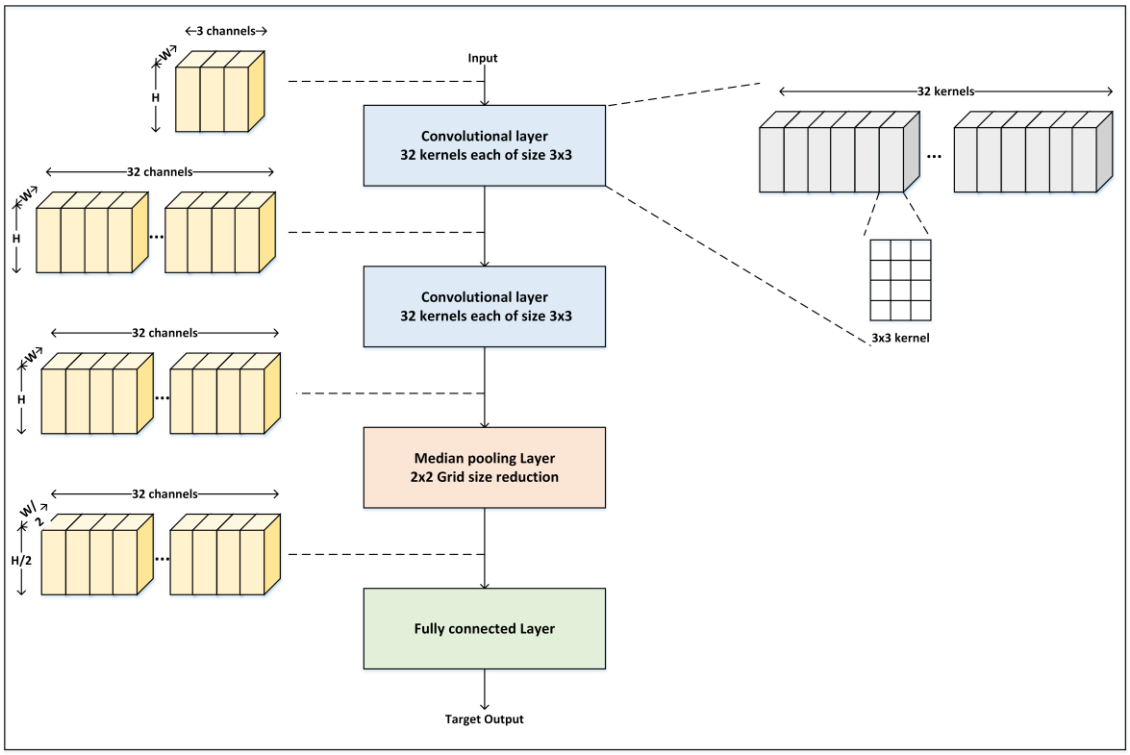

Figure 67 : Modification of the base line network to use the median layer

Table 5 shows the comparison of achieved accuracy from the baseline network and its modified versions as well as the number of increased parameters after training them for 10 epochs

Network	Top-1 error	Number of parameters/ Computation ratio
Baseline Maximum pooling Layer	34.06 %	1x
Baseline Average pooling Layer	34.2 %	1x
Modified with Median Layer	34.4%	1x

Table 5 : Comparison between Baseline network and its modified version to account for using median layer

The median layer didn't introduce any accuracy enhancement and this can be regarded to the fact that median layer is used in image de-noising problems where it is required to recover a contaminated image unlike the cifar-10 data set or any other CNN well known image classification data sets where a preprocessing step is done while collecting the images to ensure that all the images have similar distribution of intensities without any overshooting one.

Chapter 5 : Proposed Pseudo Rotated Nets

In this chapter, the generalization of the pseudo rotated kernels is proposed where full networks with different configuration are implemented demonstrating the accuracy enhancements achieved by fusing these kernels into the well-known architectures such as ResNets[29] and VGG[97] when applied on the CIFAR-10 data set[63].

5.1. ResNet Based networks

The ResNet was chosen to be the core architecture given its popularity, proven training time enhancement and the breakthrough accuracy achieved in all the ImageNet competitions. Moreover, the ResNet authors had created modified versions to experiment on CIFAR-10 data set enabling a start network that is ready for modifications as well as published results to compare with.

The start point for modification is the second version [102] where the pre activation bottleneck convolutional layer was introduced. The bottleneck layer is modified as shown in Figure 68 to account for the addition of the pseudo rotated kernels to be considered as the core layer of the network.

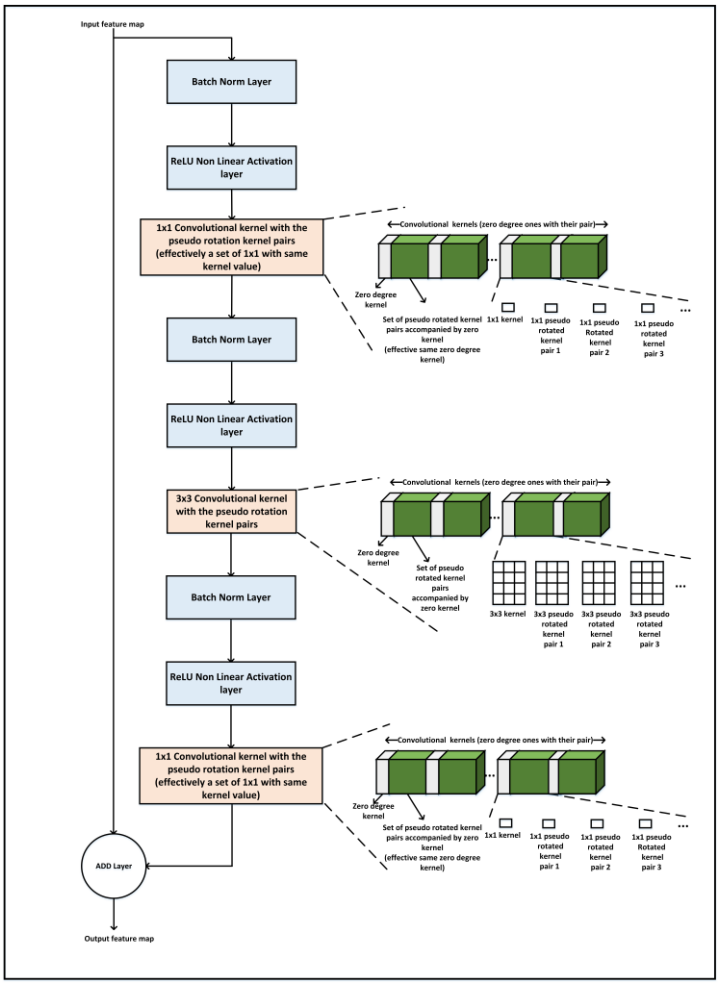

Figure 68 : Bottleneck modification for pseudo rotated kernels

This layer was modified to generate four versions one with the 180 degree rotated kernel pair as shown in Figure 69, one for 90 degree rotated kernel pairs as shown in Figure 70, one for pseudo 45 degree rotated kernels pair as shown in Figure 71 and one for one for pseudo 15 degree rotated kernels pair as shown in Figure 72. These modified layers are integrated within the ResNet full network without any modification in its structure.

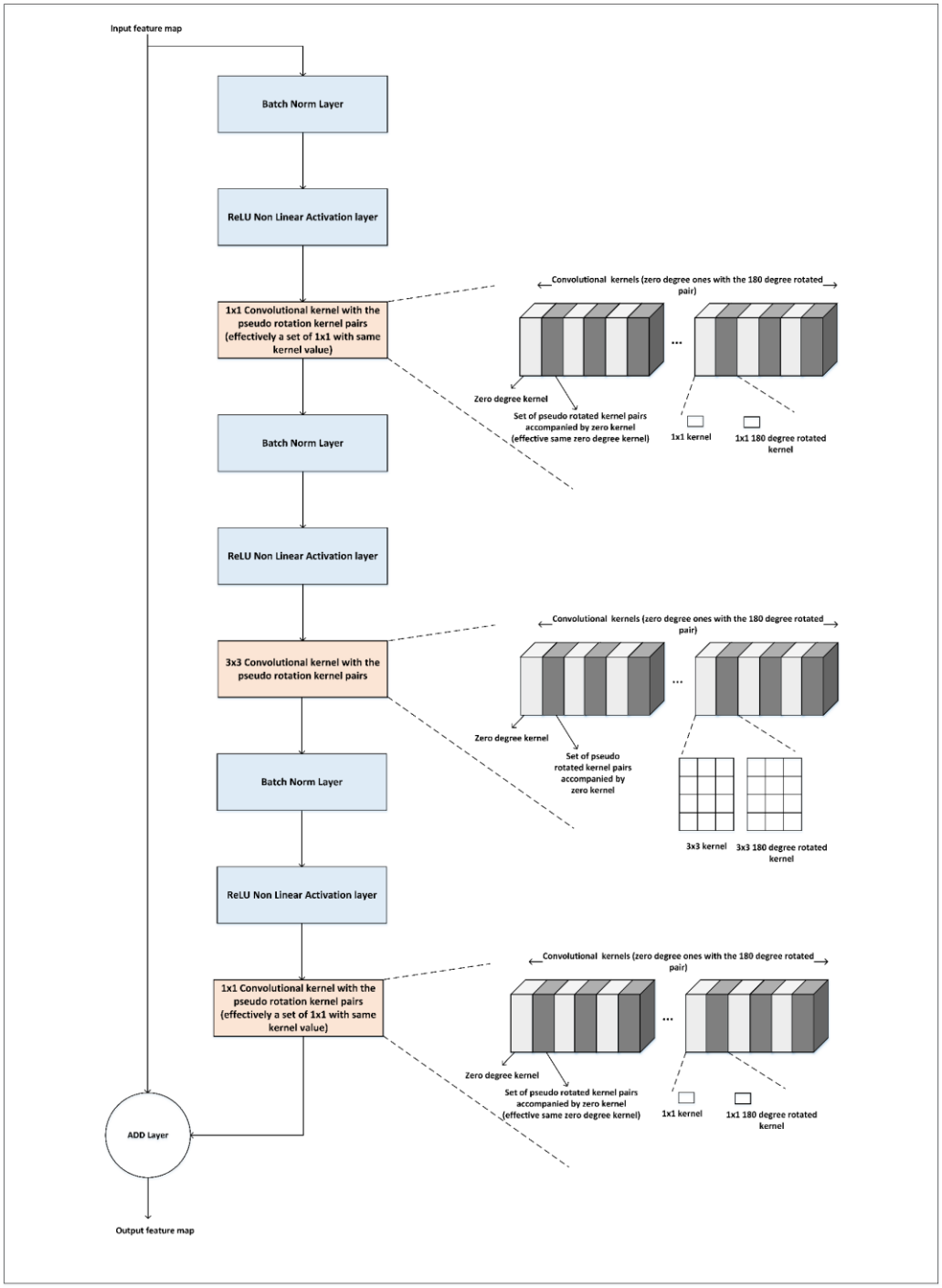

Figure 69 : Bottleneck modification for 180 degree rotated kernels

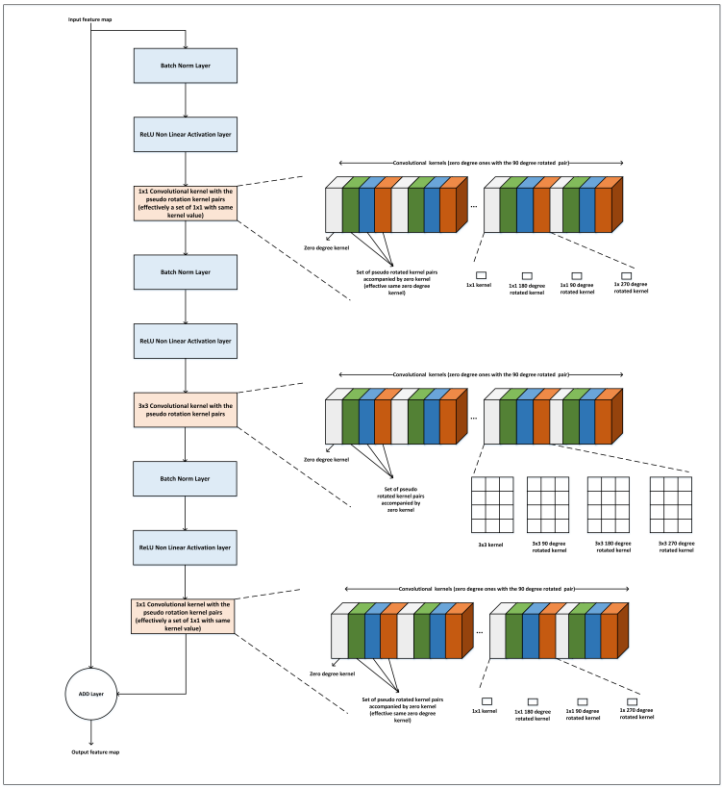

Figure 70 : Bottleneck modification for 90 degree rotated kernels

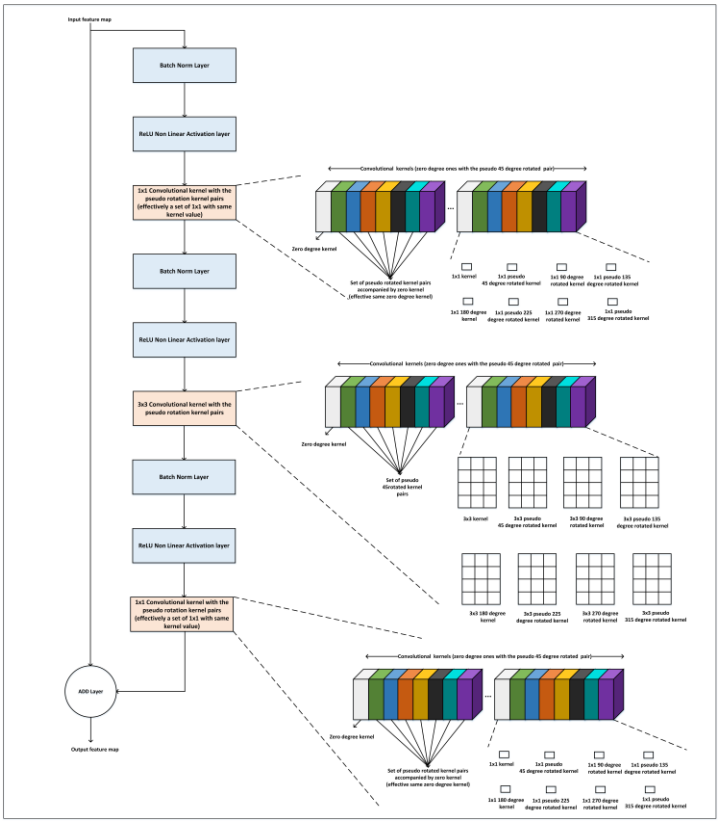

Figure 71 : Bottleneck modification for pseudo 45 degree rotated kernels

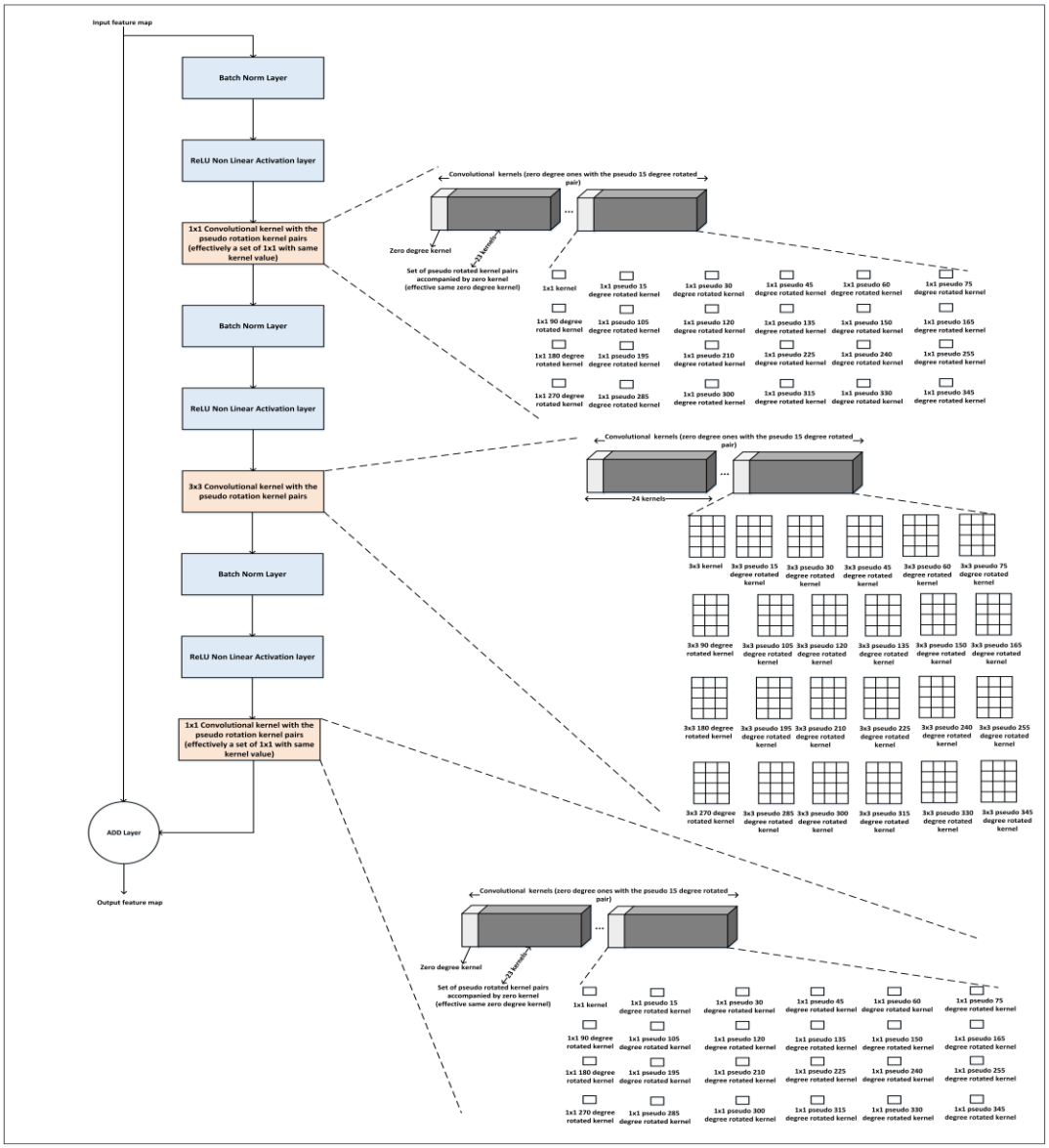

Figure 72 : Bottleneck modification for pseudo 15 degree rotated kernels

While there are many ResNet variants, the focus shall be on the ResNet-20, ResNet 56 and ResNet 110.

One note to mention here, training a full network is somehow a problematic one given the amount of computational power required which may be beyond the capability of this work. Hence, instead of training for a fixed number of epochs, the network shall be trained until reaching the accuracy saturation point where the achieved accuracy is near the published one, meanwhile increasing the number of epochs would result in minor enhancements. This would enable a fair comparison between the networks whereas the deeper networks would require more epochs to converge compared to a shallower one while maintaining a budget computational power

5.1.1. Pseudo Rotated ResNet version 1

Starting from the ResNet 110 which is one of the deepest ResNet network with around 110 layers, Table 6 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-110 and its modified versions with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house ResNet-110	N/A	1.7 Million	1x
ResNet with 180 degree kernels	N/A	3.4 Million	2x
ResNet with 90 degree kernels	N/A	6.8 Million	4x
ResNet with 45 degree kernels	N/A	13.6 Million	8x
ResNet with 15 degree kernels	N/A	40.8 Million	24x

Table 6 : Comparison between ResNet-110 and its modified pseudo rotated versions

As shown in the obtained training results such deep network was beyond the available computation infrastructure either the GPU ran out of RAM, infeasible epoch time (i.e. 10 hours) or the compiler failed to perform arithmetic optimization to fit within the GPU. Such deep networks are usually trained using a network of multiple GPUs.

Next, moving to ResNet-56 was the reasonable step where it is composed of 56 layers where Table 7 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-56 and its modified versions with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house ResNet-56	8.54 %	0.85 Million	1x
ResNet with 180 degree kernels	9.5%	1.7 Million	2x
ResNet with 90 degree kernels	8.2%	3.4 Million	4x
ResNet with 45 degree kernels	N/A	6.8 Million	8x
ResNet with 15 degree kernels	N/A	20.4 Million	24x

Table 7 : Comparison between ResNet-56 and its modified pseudo rotated versions

Again from the obtained results, some networks were shown going beyond the available computation infrastructure, however the obtained accuracy had shown either no improvements or a negligible one that can be easily claimed to be from some noise or the weight initialization. Moreover, the network didn't benefit from the added parameters and it had suffered from an over fitting problem given how fast the modified networks had reached the saturation accuracy point (nearly around the 50 or the 60 epoch). The overfitting may have occurred due to the tiny Cifar-10 image size (32x32) that doesn't require all these modifications as well as the small training data set amount, meanwhile the modifications had added a huge number of parameters.

To address this overfitting problem two architecture modifications were done. The first is to reduce the number of parameters through moving to the ResNet-20 while the second is apply the dropout [108] and spatial dropout [109] regularization techniques to help in training the increased number of parameters resulting from the modifications in the network.

Dropout is a regularization method that randomly drops out some neurons output where they are temporarily removed from the network during training. This is beneficial in terms of allowing some neuron to change in respond to the absence of some adjacent neurons to fix any unintended mistakes from other units which allows the network to be more robust. It is commonly used after the dense fully connected layers.

Spatial Dropout is an alternative regularization method that allows similar dropout manner to be applied within the convolutional neural networks whereas an entire channel is dropped from the feature map within the convolutional layer in analogous to how the neurons are dropped in the fully connected layers.

Another choice was made given the limited computational power available is to choose the pseudo 45 degree rotated kernels as the pair of kernels used within the bottleneck modified module.

All the aforementioned choices had led to the proposal of the Pseudo Rotated ResNets version 1 which is shown in Figure 73 and its associated pseudo 45 degree rotated kernels bottleneck layer modification to account for spatial dropout is shown in Figure 74

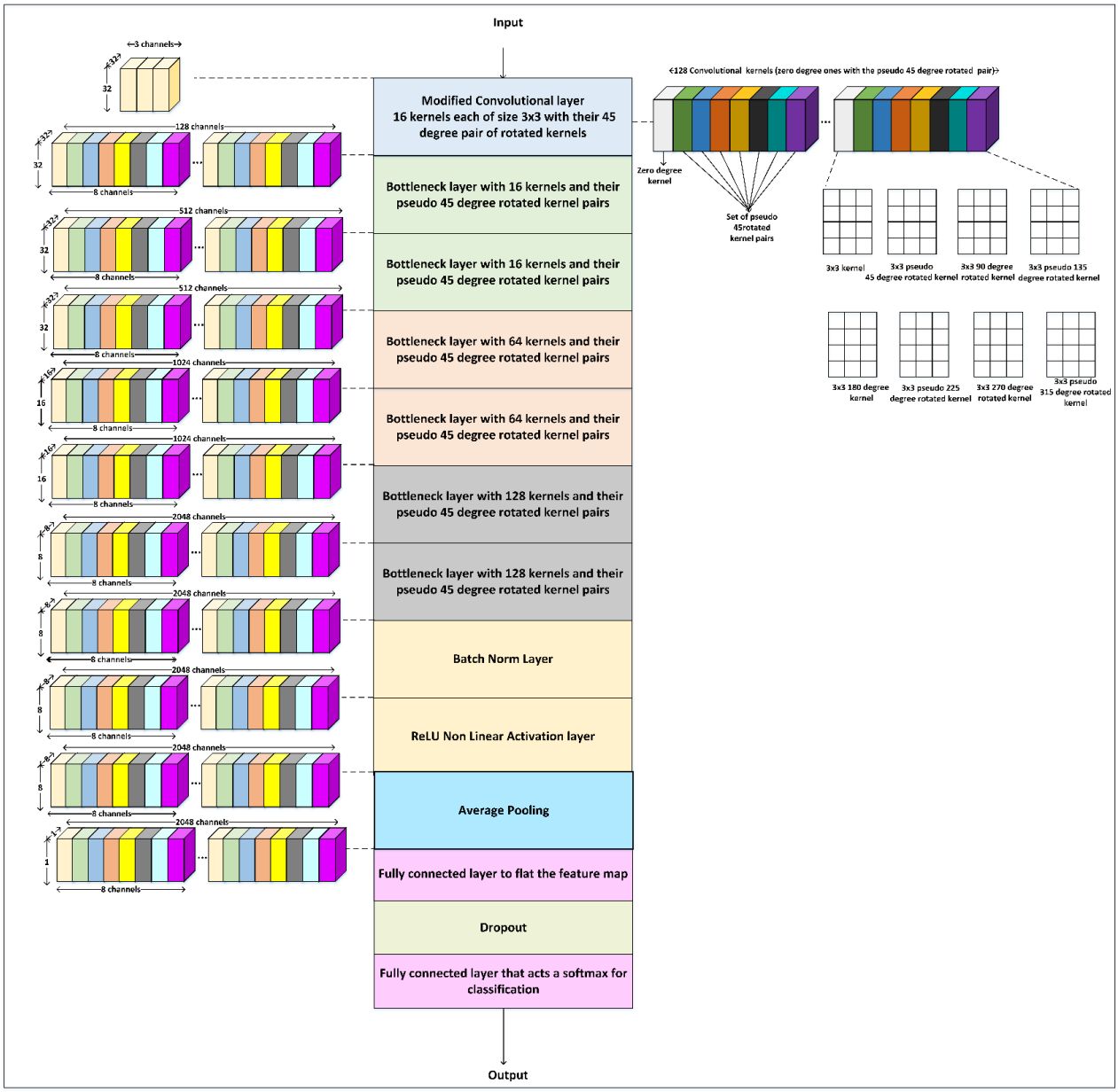

Figure 73 : Pseudo Rotated ResNet version 1

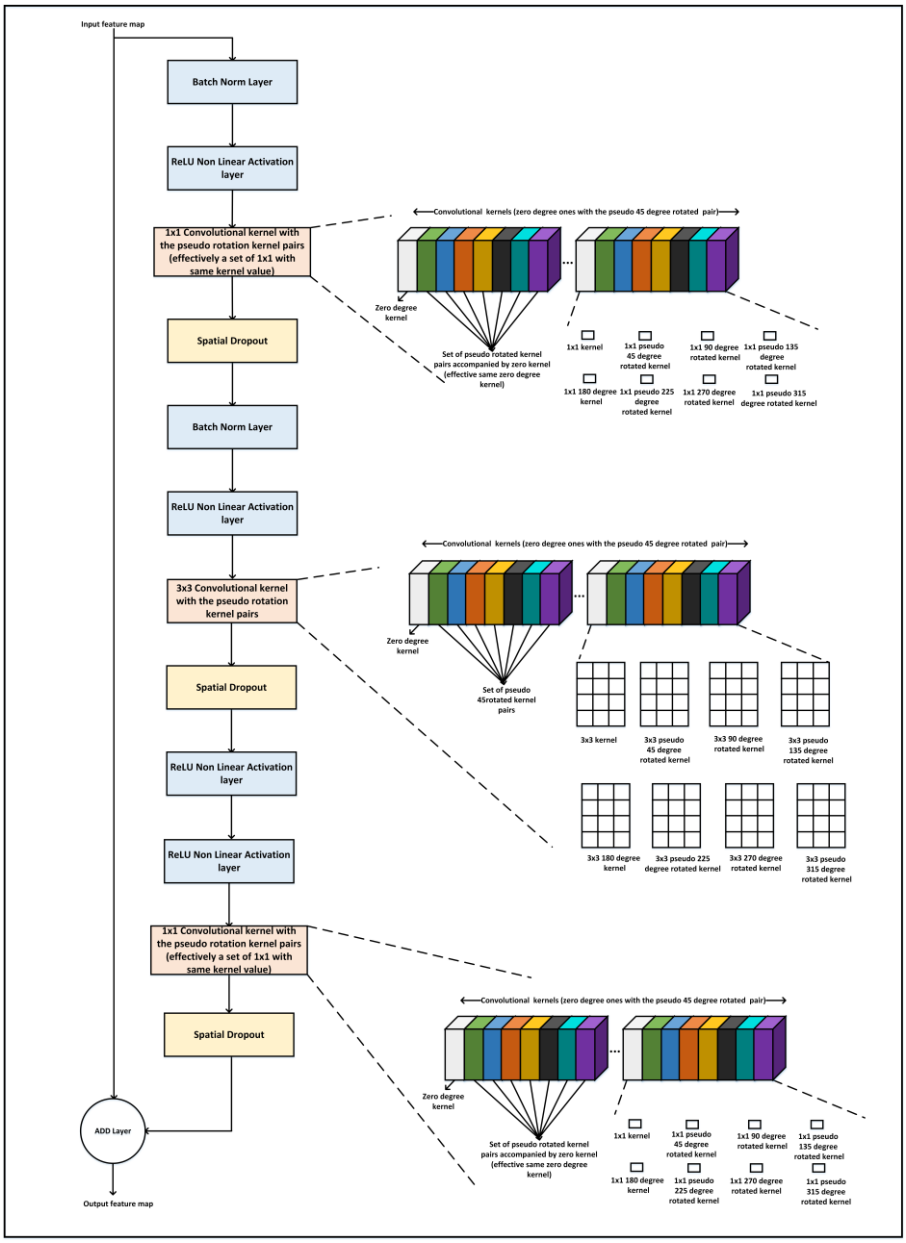

Figure 74: Pseudo 45 degree rotated kernels bottleneck with spatial dropout

Table 8 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-20, in house implemented ResNet-56 and the Pseudo Rotated ResNet version 1 with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house ResNet-20	9.6 %	0.27 Million	1x
In house ResNet-56	8.54 %	0.85 Million	3.1x
Pseudo Rotated ResNet version 1	7.1 %	2.16 Million	8x

Table 8 : Comparison between ResNet-20, ResNet-56 and Pseudo Rotated ResNet version 1

The results obtained shows that the network started to benefit from the pseudo rotated kernel pairs attached to the convolution layers, meanwhile the modified network size and the associated added regularization methods had resolved some of the overfitting problem resulting in an accuracy improvement. However, this accuracy enhancement had come with an increase in number of parameters which shall be addressed in version 2.

5.1.2. Pseudo Rotated ResNet version 2

The enhanced accuracy shows the potential of the pseudo rotated kernels, however it comes with a penalty in terms of increased number of parameters.

To address this increase, the circle space of the pseudo rotated kernels described before in the previous chapter was revisited where it is required to search for another pseudo rotated kernels combination that maintain the achieved accuracy meanwhile reducing the number of parameters.

A useful insight here is to consider reducing the number of kernels within the pseudo 45 degree rotated kernels set where it is proposed to reduce the pseudo 45 degree rotated kernels pairs to exclude the 90 degree multiples and shall be noted as pseudo 45 degree without 90 corners as shown in Figure 75.

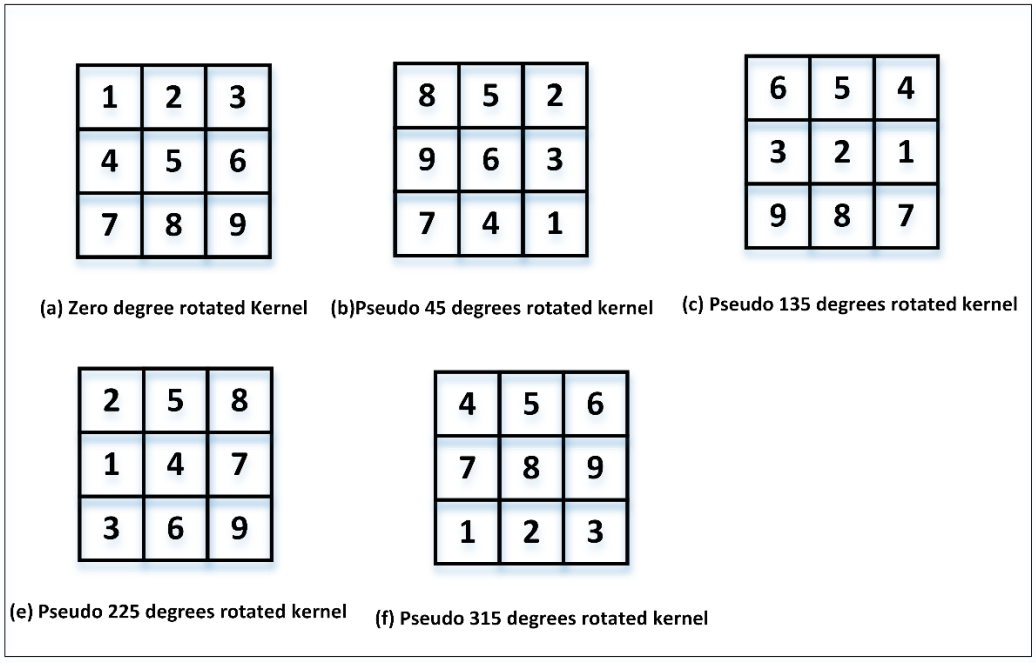

Figure 75 : Pseudo 45 degree without 90 corners

This will result in modification to the bottleneck layer as shown in Figure 76.

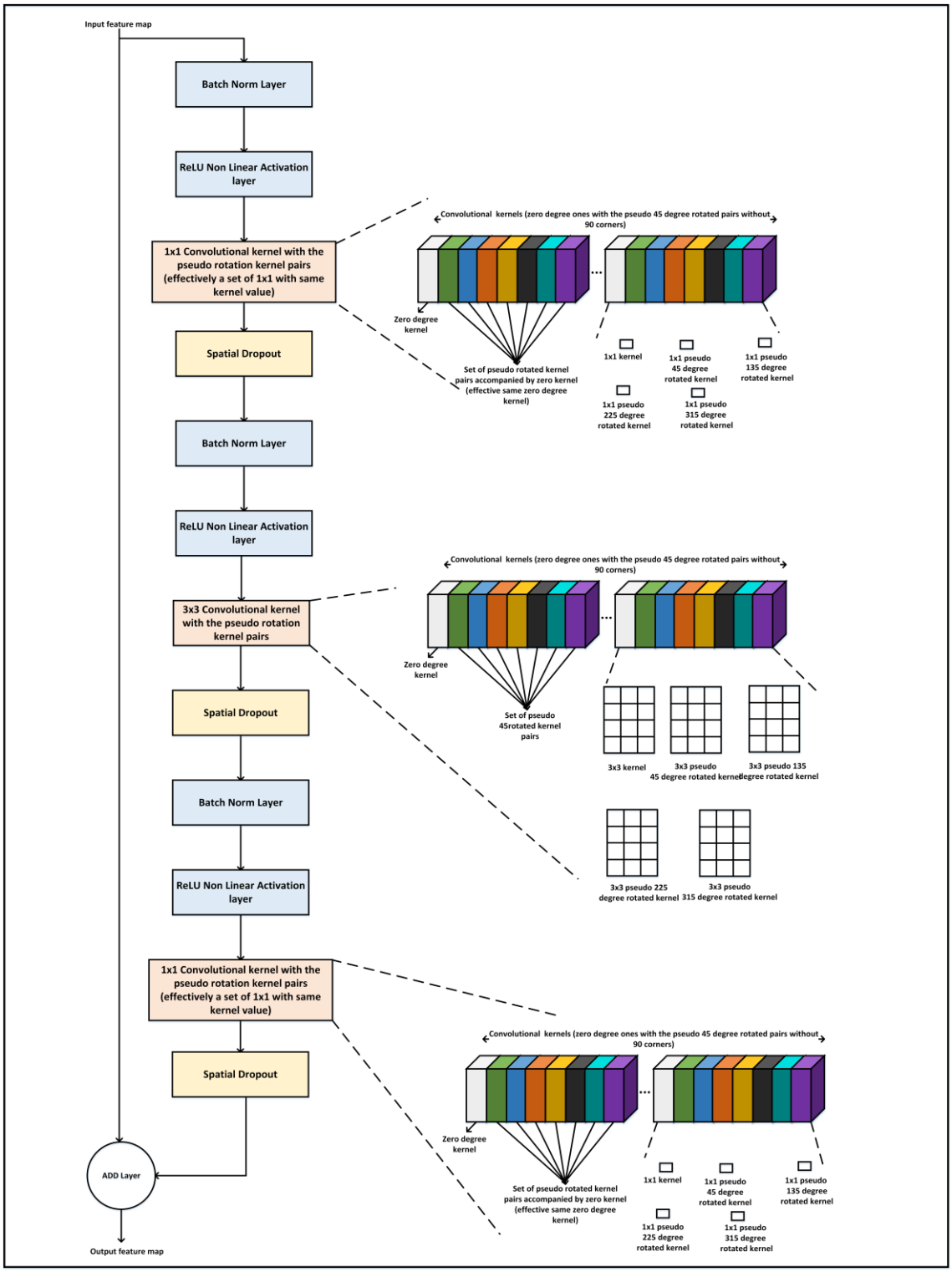

Figure 76 : Pseudo 45 degree rotated kernels without 90 corners bottleneck

However, this will come with the cost of removing the 180 degree rotated kernel which was the one responsible for the translation property enhancement. To mitigate that loss, a modification to the network structure was done where the first and second set of the convolutional layers shall use the introduced pseudo 45 degree without 90 corners, meanwhile the third set shall use the 180 degree rotated kernels only.

The Intuition here is that the first and second sets of convolutional layers shall benefit from the enhanced rotation property while going near the end of the network where the features becomes more expressiveness and thus enhancing the translation property shall be beneficial. This is basic idea behind the Pseudo Rotated ResNet version 2 which is shown in Figure 77

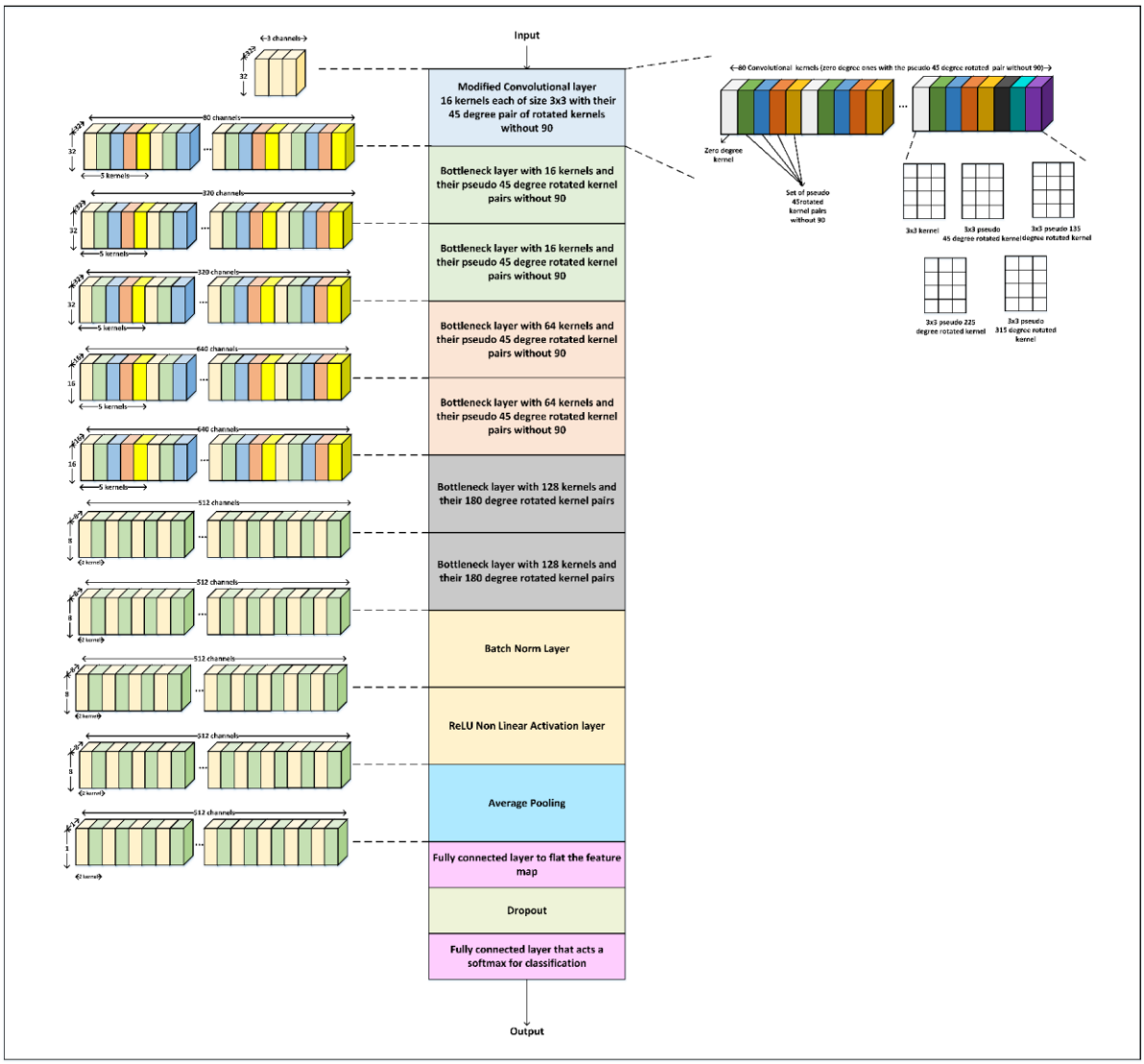

Figure 77 : Pseudo Rotated ResNet version 2

Table 9 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-20, in house implemented ResNet-56, the Pseudo Rotated ResNet version 1 and version 2 with their total number of parameters as well as the number of increased parameters.

Network	**Top-1 error**	**Total Number of parameters**	**Parameters increase ratio**
In house ResNet-20	9.6 %	0.27 Million	1x
In house ResNet-56	8.54 %	0.85 Million	3.1x

Pseudo Rotated ResNet version 1	7.1 %	2.16 Million	8x
Pseudo Rotated ResNet version 2	6.08 %	0.83 Million	3.07x

Table 9 : Comparison between ResNet-20, ResNet-56, Pseudo Rotated ResNet version 1 and version 2

The results obtained show that the parameters were reduced to be comparable with the ResNet-56 with an enhanced accuracy even more than Pseudo Rotated ResNet version 1 showing that increasing the width using the pseudo rotated kernels may be more performance beneficial rather than increasing the depth. The enhancement in the accuracy while decreasing the number of parameters can be regarded to reducing the overfitting by decreasing the number of parameters carefully through the distribution of the more rotating kernels at the first stages while focusing on enhancing the translation at the later ones.

5.1.3. Pseudo Rotated ResNet version 3

After showing the capability of the pseudo rotated kernels to enhance the accuracy with a reasonable number of parameters, it is required to squeeze the network more in attempt to boost the accuracy performance.

Reviewing back the affine transformation properties, one property seems to be interesting is the scaling one. Successively applying the scaling property can push the network one more step towards being capable to unify more properties of the affine transformations within its processing.

Scaling can be done with the most straight forward approach through applying the pooling techniques as an attached kernel within the convolutional kernels. However, this direct apply of the pooling techniques could lead to the explode of parameters number given it maintains the same number of channels from the previous layer feature map while it scales its height and width. For instance, as shown in Figure 78, in the Pseudo Rotate ResNet version 2 first layer outputs 80 channels within the generated feature map, if the pooling layer is directly applied in the next one it would generate 80 channels in the output feature map in addition to the 320 channels generated from the already existing convolutional kernels which would result in a total 400 channels in the final feature map.

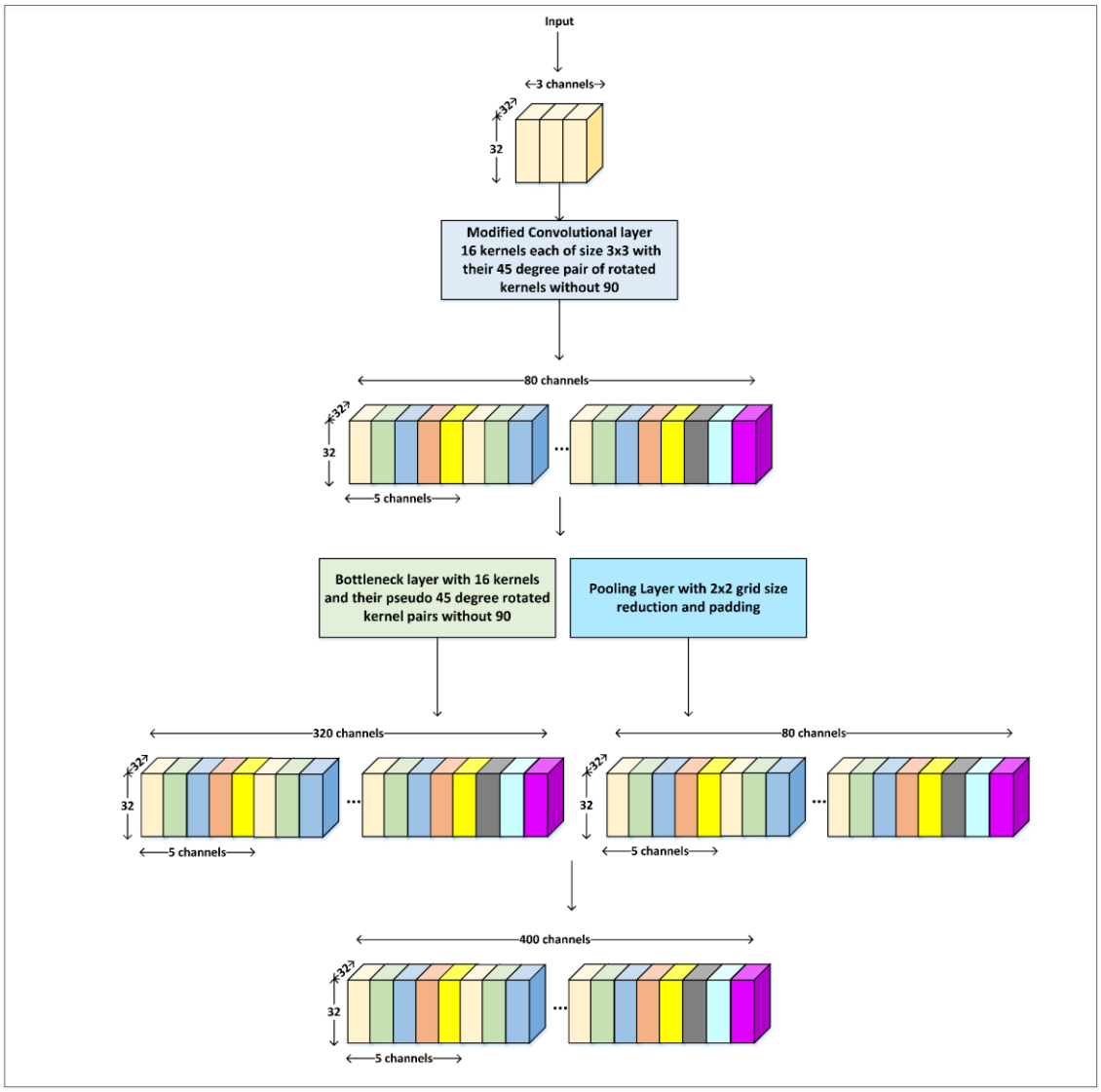

Figure 78 : Direct apply of pooling layer within the first layer Pseudo Rotated version 2

Moving with this approach across different layers would lead to an exponential growth in number of parameters which would go beyond the computational power budget. Analyzing the exploding number of parameters, it seems it is required to reduce the number of channels when applying the pooling techniques.

Inspired from the Inception module where the applied pooling layers are followed by a 1x1 convolution to reduce the number of channels and hence reduces the number of parameters, it seems reasonable to follow their footsteps and apply the same approach where each pooling technique shall be followed by a 1x1 convolutional kernel.

Thus, from the aforementioned, the bottleneck module is modified as shown in Figure 79 to add maximum pooling kernel with 2x2 grid size reduction configured to allow the padding method to keep the generated feature map dimension similar to the input one enabling its further concatenation with the feature maps generated from the convolutional kernels, meanwhile it is followed by 1x1 convolutional kernel also to allow parameters reduction.

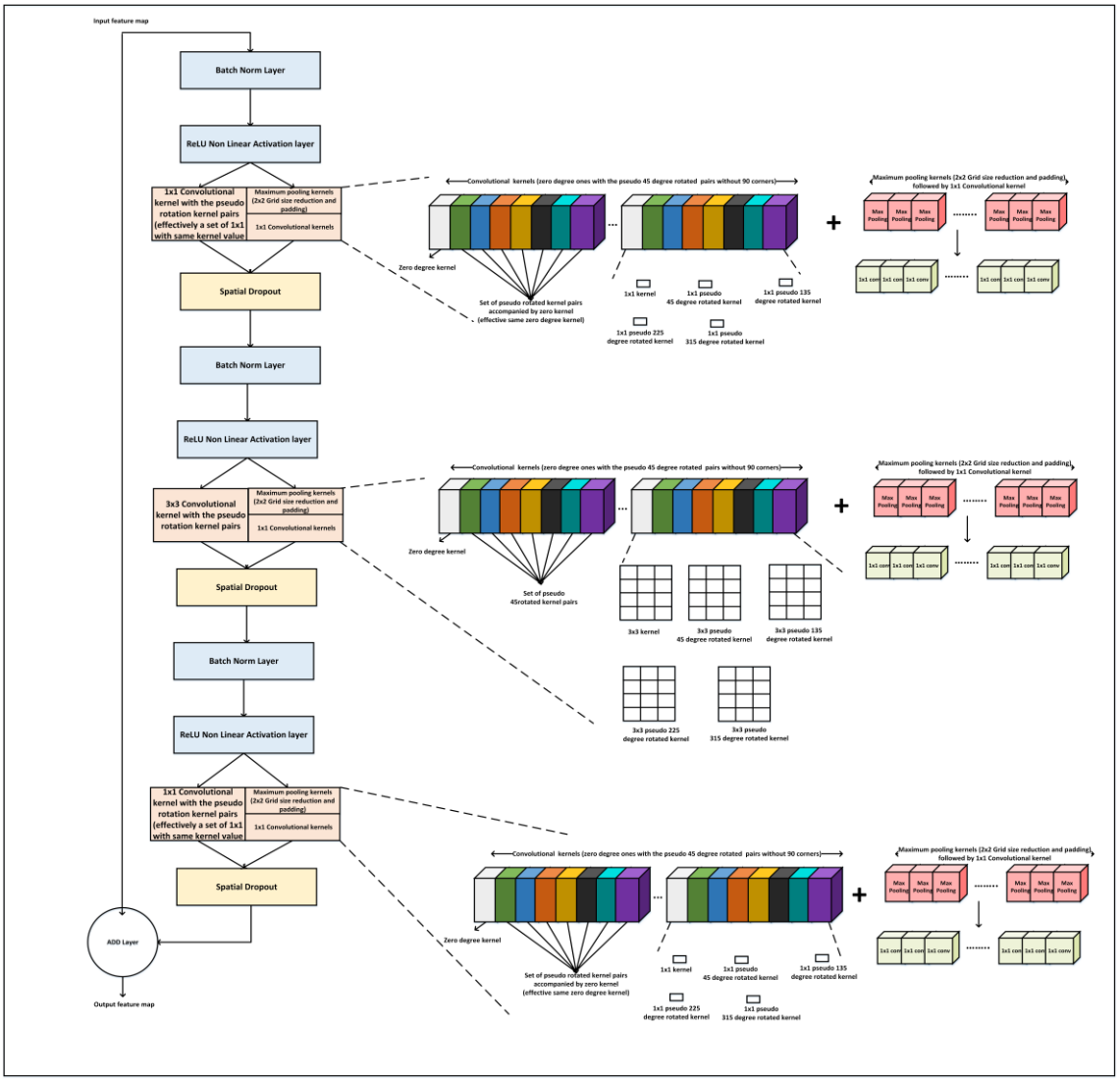

Figure 79 : Pseudo 45 degree rotated kernels without 90 corners bottleneck with an additional maximum pooling kernels

The network structure remains the same as pseudo Rotated ResNet version 2 as shown in Figure 80.

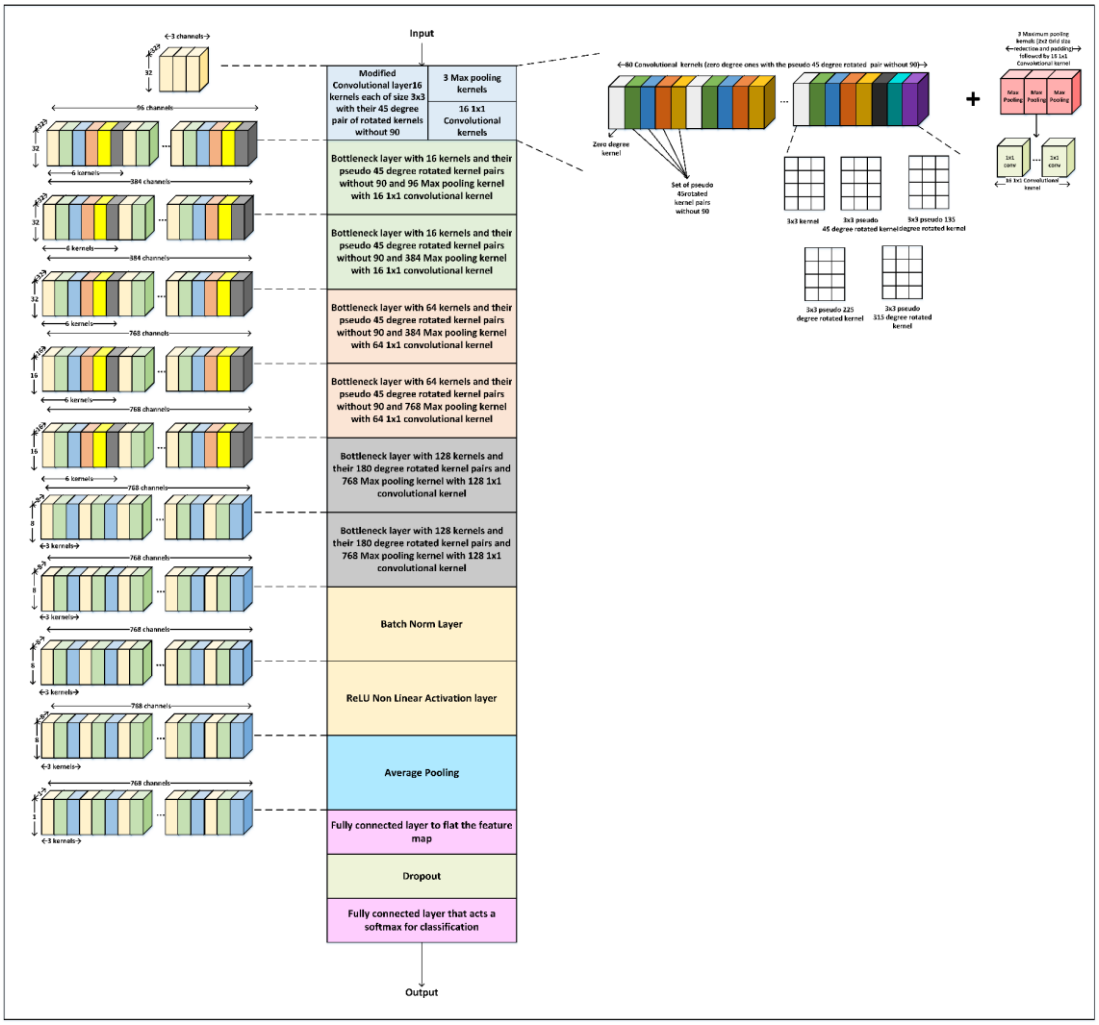

Figure 80: Pseudo Rotated ResNet version 2 with additional maximum pooling kernels modification

Table 10 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-20, in house implemented ResNet-56, the Pseudo Rotated ResNet version 1, version 2 and the modified version 2 to include the maximum pooling kernel with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house ResNet-20	9.6 %	0.27 Million	1x
In house ResNet-56	8.54 %	0.85 Million	3.1x
Pseudo Rotated ResNet version 1	7.1 %	2.16 Million	8x
Pseudo Rotated ResNet version 2	6.08 %	0.83 Million	3.07x
Modified Pseudo Rotated ResNet version 2 with maximum pooling kernel	5.5%	1.61 Million	5.65x

Table 10 : Comparison between ResNet-20, ResNet56, Pseudo Rotated ResNet versions 1 and 2 as well as Pseudo Rotated ResNet versions 2 with maximum pooling

The results showed some improvement from applying the maximum pooling in attempt to achieve the scaling property of the affine transformation.

A greedy approach is to consider adding the average pooling in a similar manner to the maximum pooling to boost the scaling property more. Thus, the bottleneck module is modified as shown in Figure 81. to add the average pooling kernel with same configuration as maximum pooling and also shall be followed by 1x1 convolutional kernel.

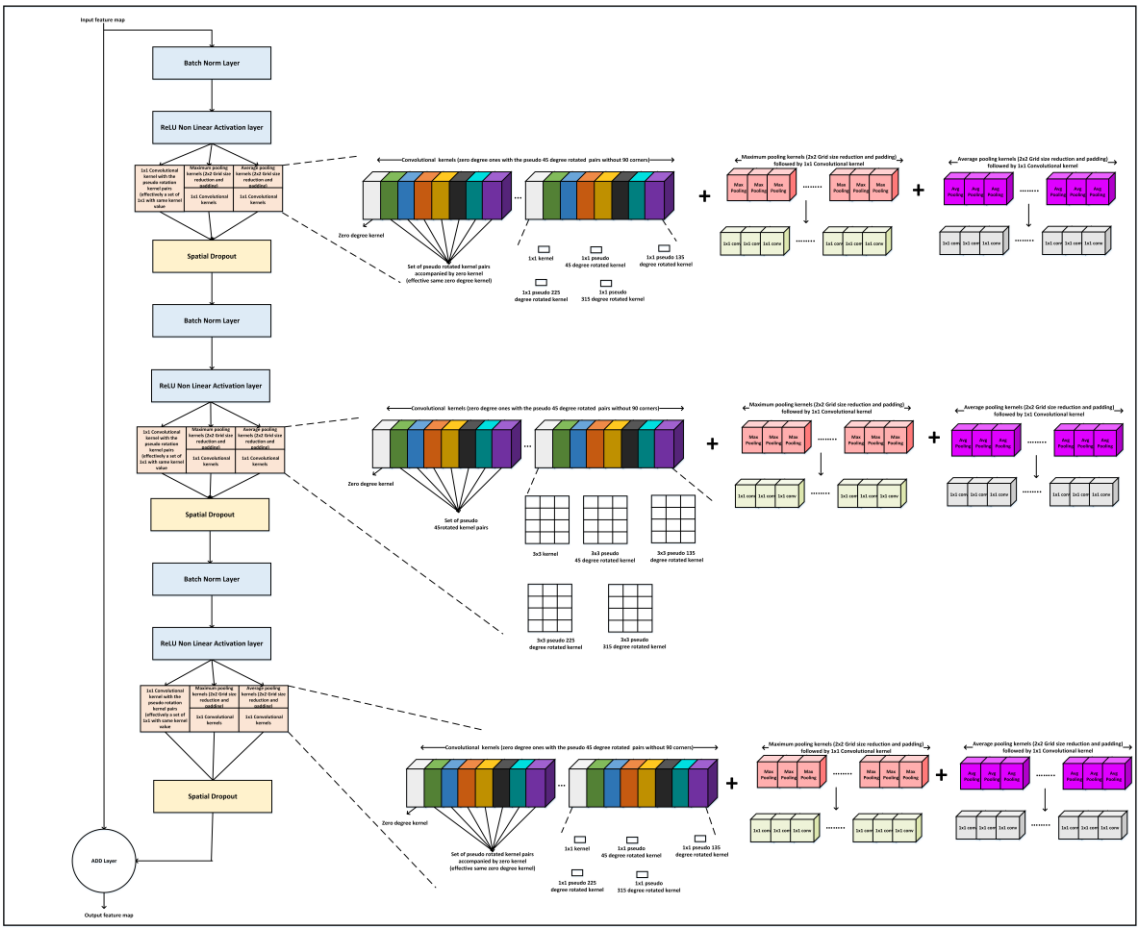

Figure 81: Pseudo 45 degree rotated kernels without 90 corners bottleneck with an additional maximum and average pooling kernels

An insight here is that one can rethink that the network is approaching to be self-augmented where the basic augmentation techniques such as rotation and scaling are already done within the network only adding noise is the missing basic technique. Thus, another modification is done to apply Gaussian noise at the input image before passing through the network.

All the aforementioned had led to the introduction of Pseudo Rotated ResNet version 3 which is shown in Figure 82

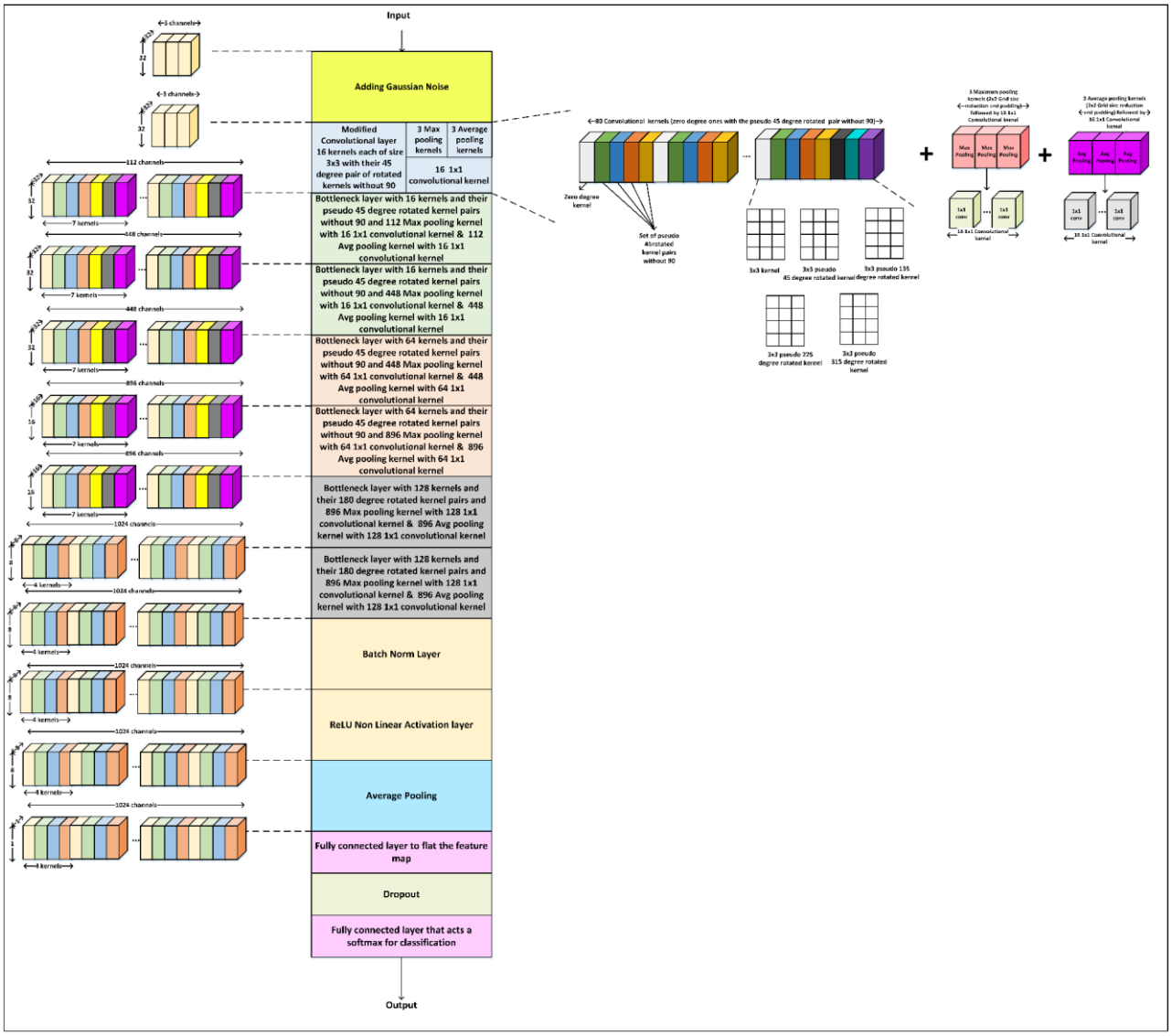

Figure 82 : Pseudo Rotated ResNet version 3

Table 11 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house implemented ResNet-20, in house implemented ResNet-56, the Pseudo Rotated ResNet version 1, version 2, the modified version 2 and version 3 with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house ResNet-20	9.6 %	0.27 Million	1x
In house ResNet-56	8.54 %	0.85 Million	3.1x
Pseudo Rotated ResNet version 1	7.1 %	2.16 Million	8x
Pseudo Rotated ResNet version 2	6.08 %	0.83 Million	3.07x

Modified Pseudo Rotated ResNet version 2 with maximum pooling kernel	5.5%	1.61 Million	5.65x
Pseudo Rotated ResNet version 3	4.7 %	2.658 Million	9.3 x

Table 11 : Comparison between ResNet-20, ResNet56 and different Pseudo Rotated ResNet versions

The obtained results show an improvement in the accuracy where the network benefited from the added average and maximum pooling layers without overfitting. However, this improvement had increased the number of parameters significantly. Addressing this increase would require revisiting the circle space of the pseudo rotated kernels or even revisiting the way the pooling layers were attached to the network which is left to future work.

5.2. VGG Based networks

VGG is one of the widely adopted CNN given its symmetric architecture and the straight forward structure.

Generalizing on the VGG was a necessary step to demonstrate how the pseudo rotated kernels can be applied in different architectures leading to accuracy enhancements.

Unfortunately, VGG wasn't applied on the CIFAR-10 data set, thus choosing, creating and modifying the network was done from scratch.

Given how giant is the network compared to the tiny data set used and to allow network training to be within the available computational budget, VGG-11 architecture was selected with three modifications to the structure. The first is to adjust all the convolutional kernels to match the CIFAR-10 images dimensions rather than the ImageNet one, while the second was adding the Batch Normalization layer after each convolutional one, meanwhile the third was adding spatial dropout between every two consecutive convolutional layers. These modifications were required in attempt to regularize this data hungry network as well as accelerating the training procedure. The modified VGG-11 can be shown in Figure 83 and Figure 84

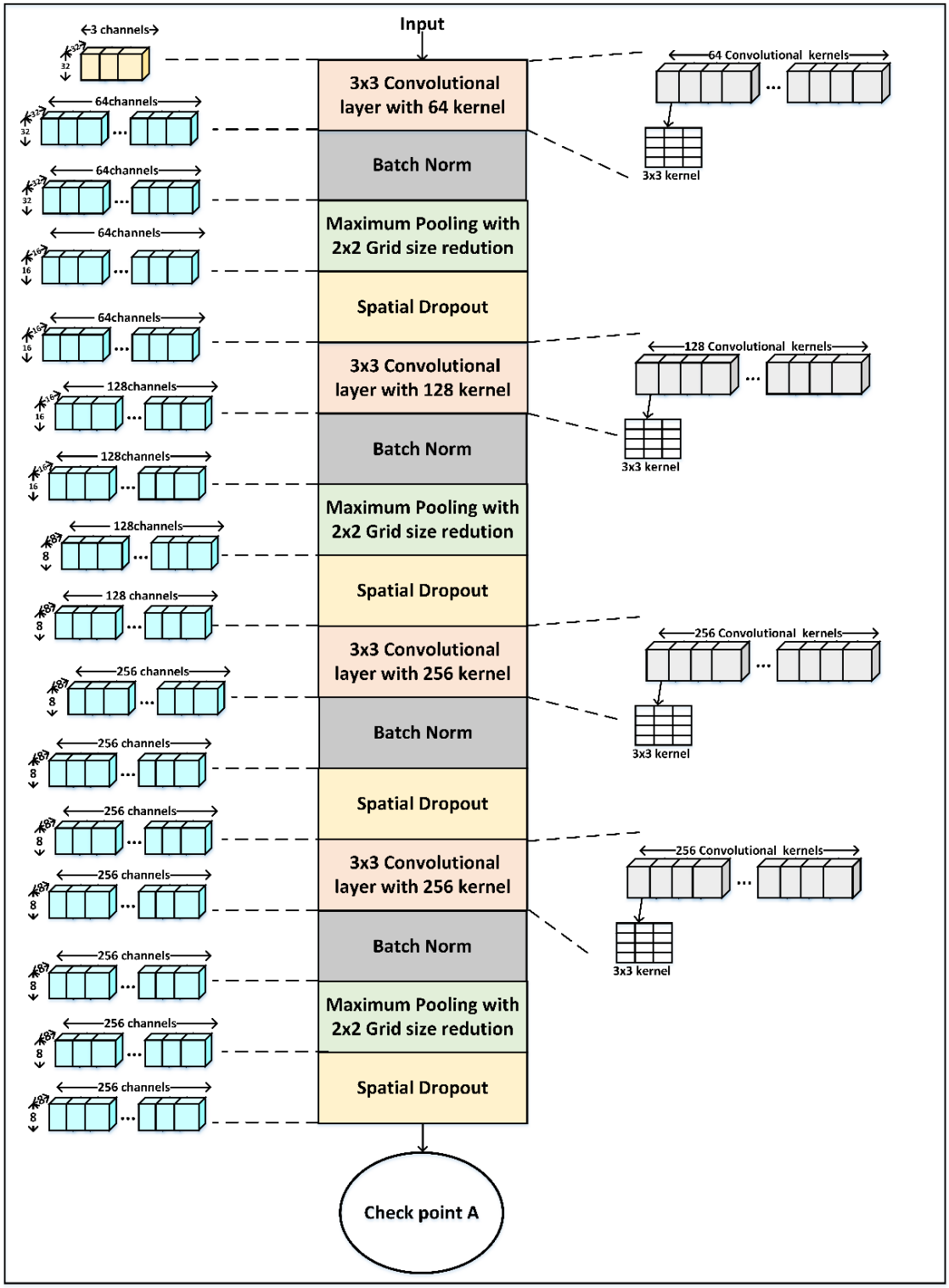

Figure 83 : Modified Baseline VGG-11 Part A

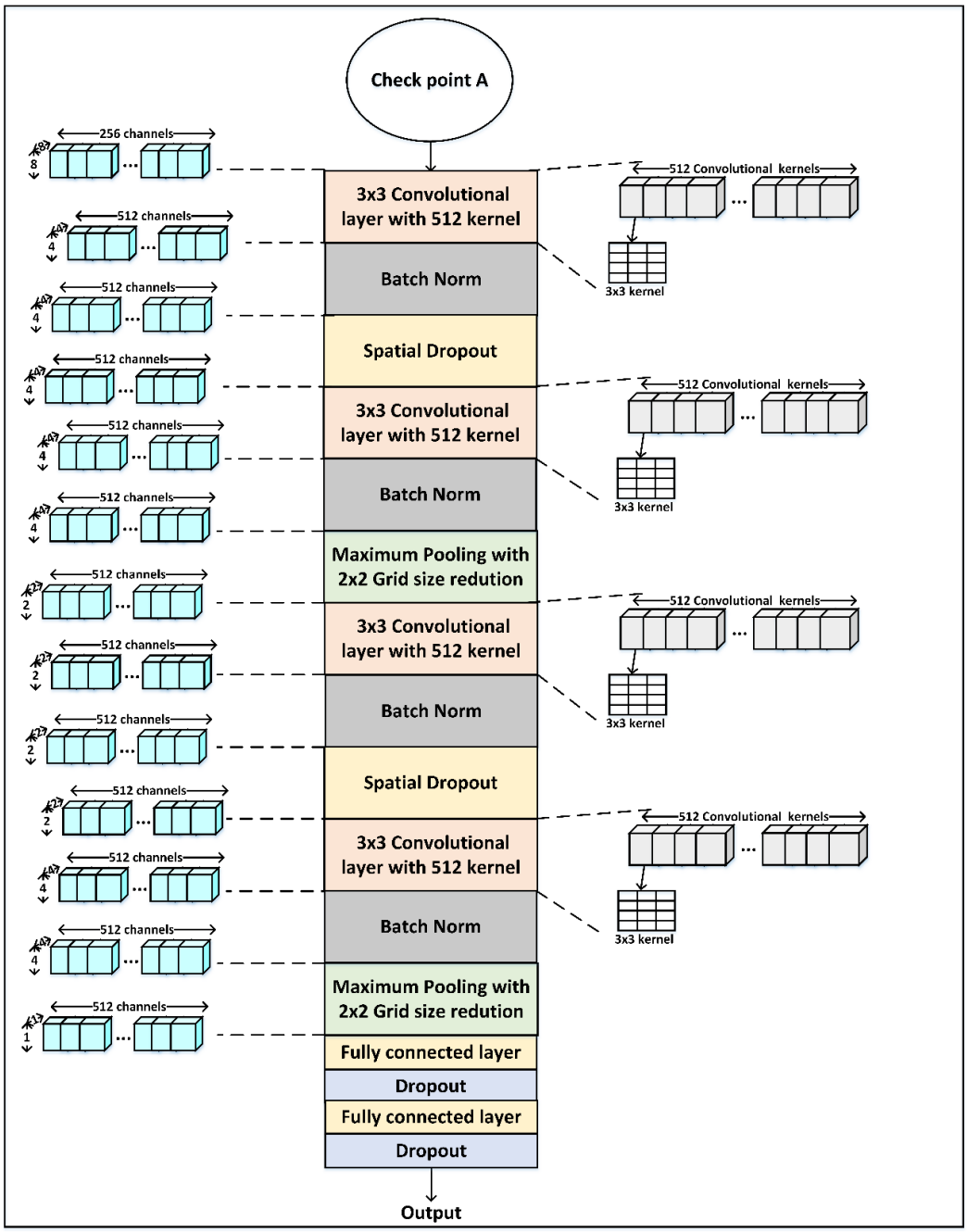

Figure 84 : Modified Baseline VGG-11 Part B

5.2.1. Pseudo Rotated VGG version 1

This version shall follow the footsteps of the pseudo Rotated ResNet version 2 where the first three stacks of the convolutional kernels shall be modified to have the pseudo 45 degree rotated kernels without 90 corners pairs while the last two stacks shall be modified to have the 180 degree rotated kernel pairs. The Pseud Rotated VGG version 1 can be shown in Figure 85 and Figure 86

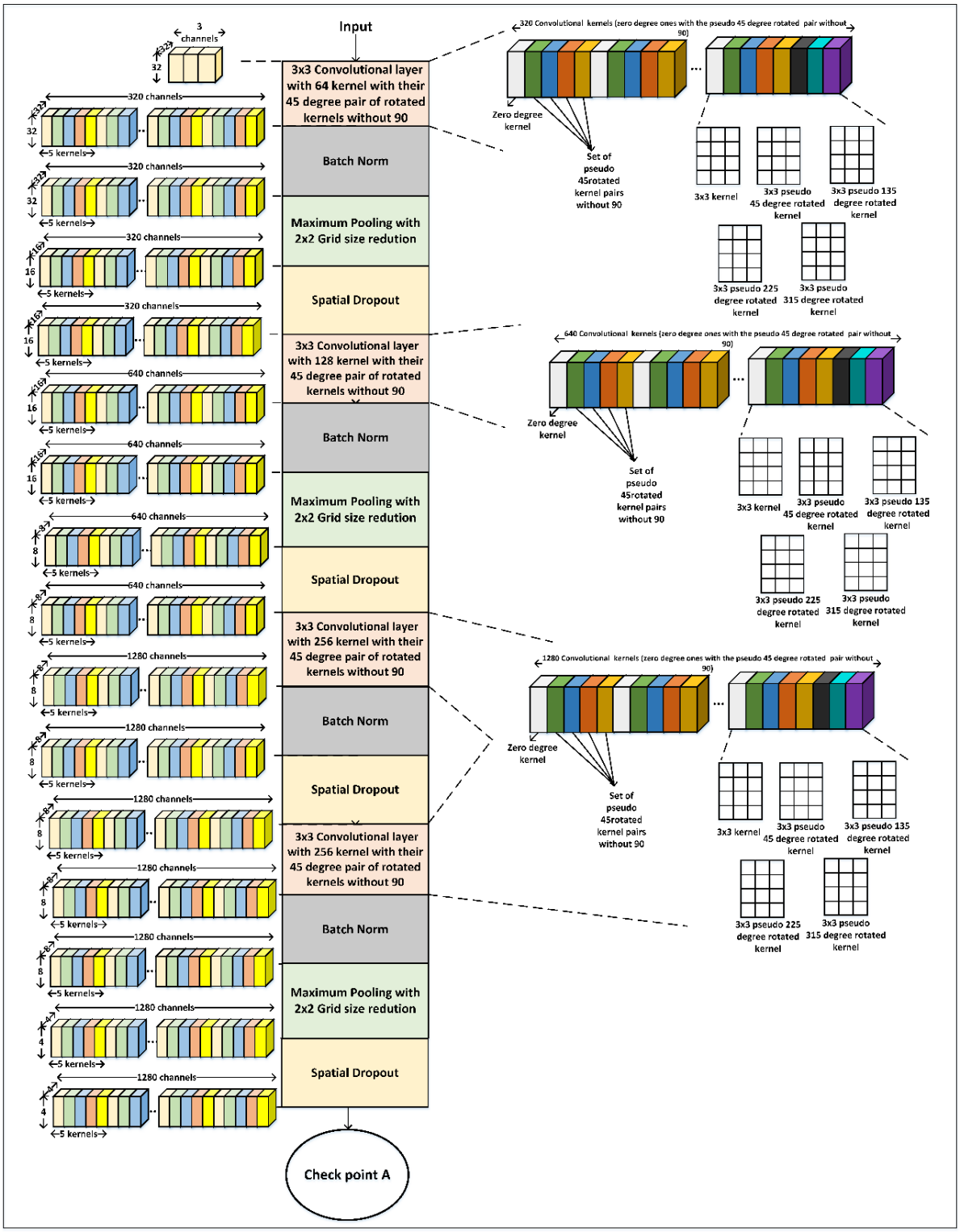

Figure 85: Pseudo Rotated VGG version 1 Part A

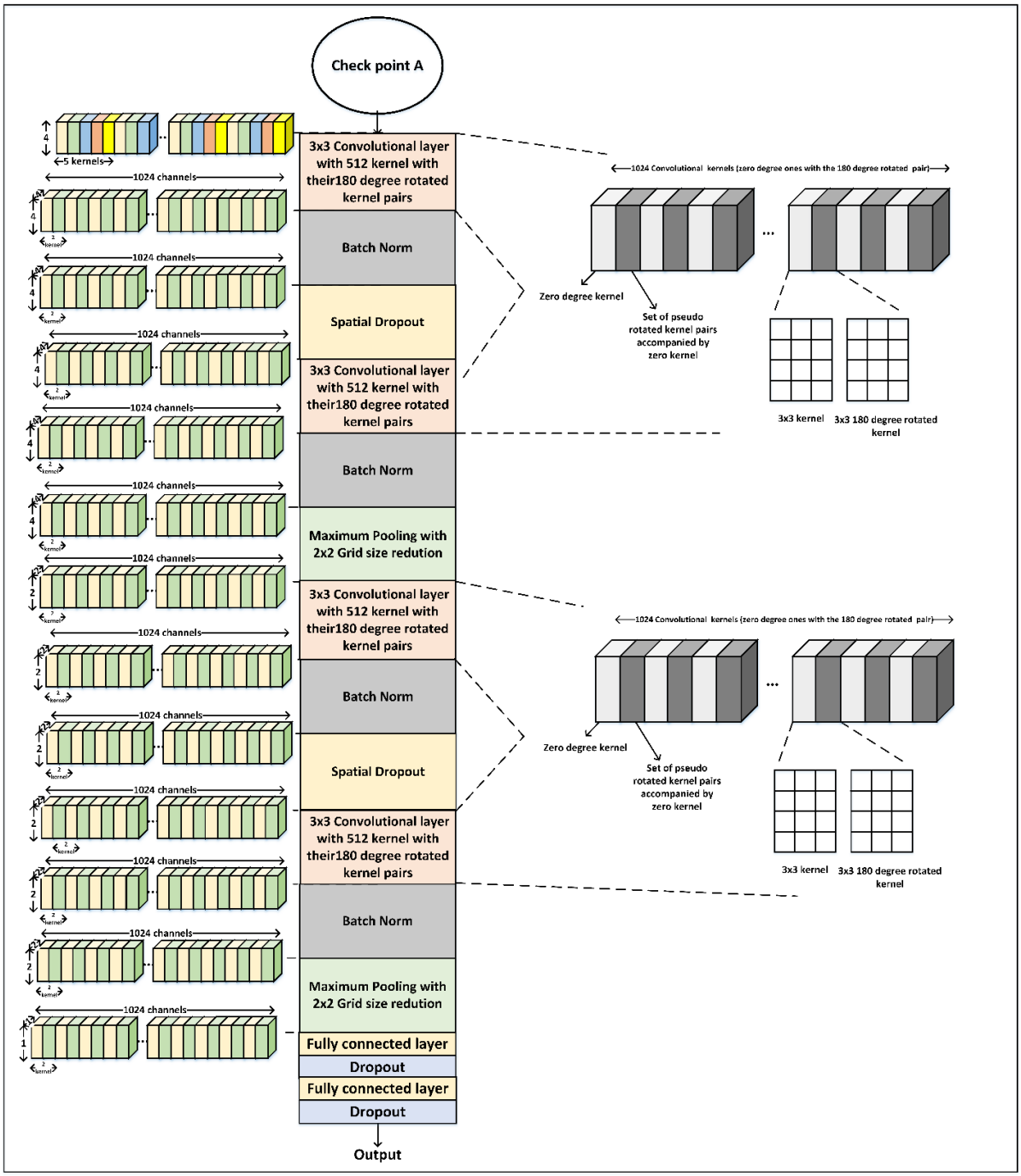

Figure 86: Pseudo Rotated VGG version 1 Part B

Table 12 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house modified VGG-11and Pseudo Rotated VGG version 1 with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house modified VGG-11	9.5 %	24,149,519 Million	1x

Pseudo Rotated VGG version 1	7.15 %	45,886,730 Million	1.9x

Table 12 : Comparison between modified VGG-11 and Pseudo Rotated VGG version 1

The obtained results show an improvement in the accuracy demonstrating how the pseudo rotated kernels had generalized to be successively fused within the VGG network. Moreover, the increase in the number of parameters, didn't introduce much overfitting showing how the network had benefited from the added parameters

5.2.2. Pseudo Rotated VGG version 2

This version shall follow the footsteps of the pseudo Rotated ResNet version 3 in attempt to generalize the self-augmented network idea as well as the movement towards a unified affine transformation.

The pseudo Rotated VGG version is modified to account for adding Gaussian noise to the input image as well as extending the convolutional kernels to account for maximum and average pooling kernels with their subsequent 1x1 convolutional kernels.

Figure 87, Figure 88 and Figure 89 introduce the Pseudo Rotated VGG version 2 in accordance to the aforementioned modifications.

Table 13 shows the comparison of achieved accuracy after training for the accuracy saturation point defined earlier from the in house modified VGG-11and Pseudo Rotated VGG version 1 and version 2 with their total number of parameters as well as the number of increased parameters.

Network	Top-1 error	Total Number of parameters	Parameters increase ratio
In house modified VGG-11	9.5 %	24,149,519 Million	1x
Pseudo Rotated VGG version 1	7.15 %	45,886,730 Million	1.9x
Pseudo Rotated VGG version 2	5.8 %	78,147,338 Million	3.2x

Table 13 : Comparison between modified VGG-11 and Pseudo Rotated VGG versions

The obtained results show an improvement in the accuracy demonstrating that stretching the network capability to have a built in rotating, scaling and enhanced translation properties would be fruitful from a performance perspective.

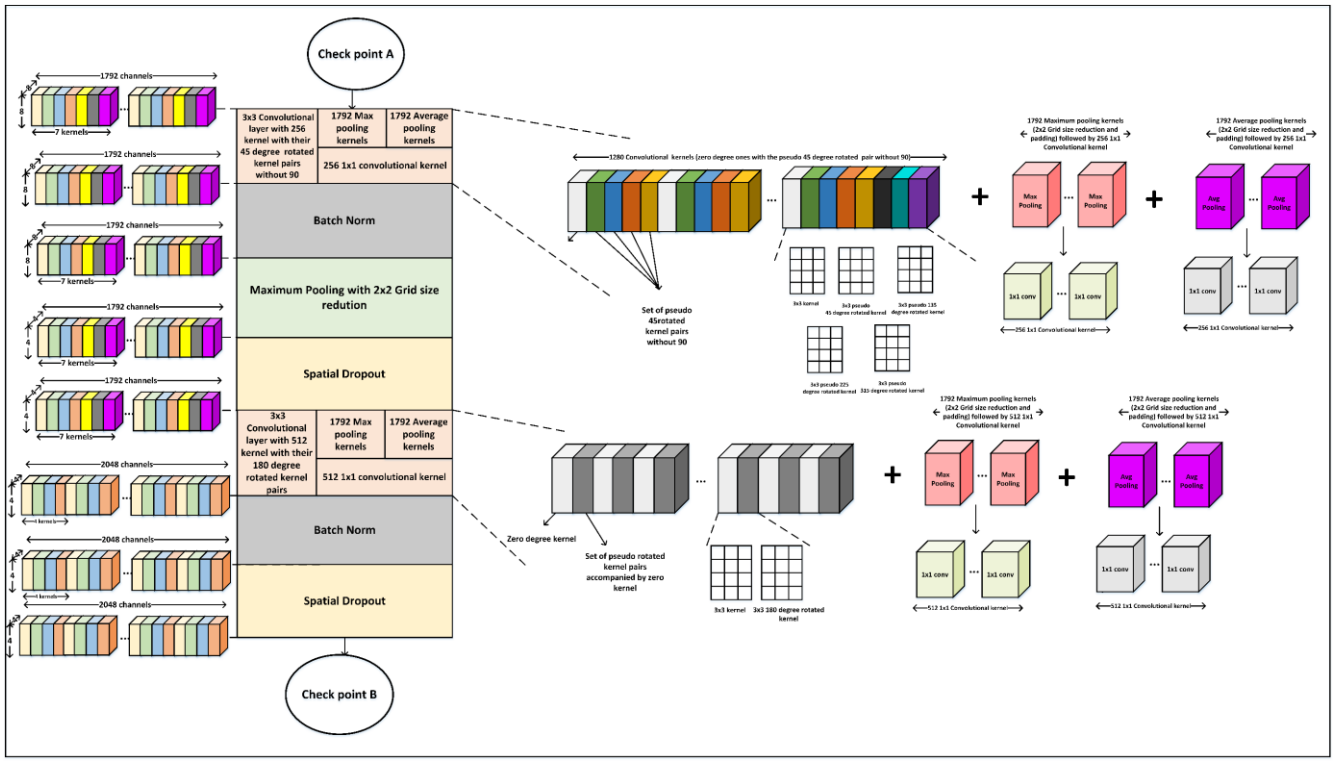

Figure 87: Pseudo Rotated VGG version 2 Part A

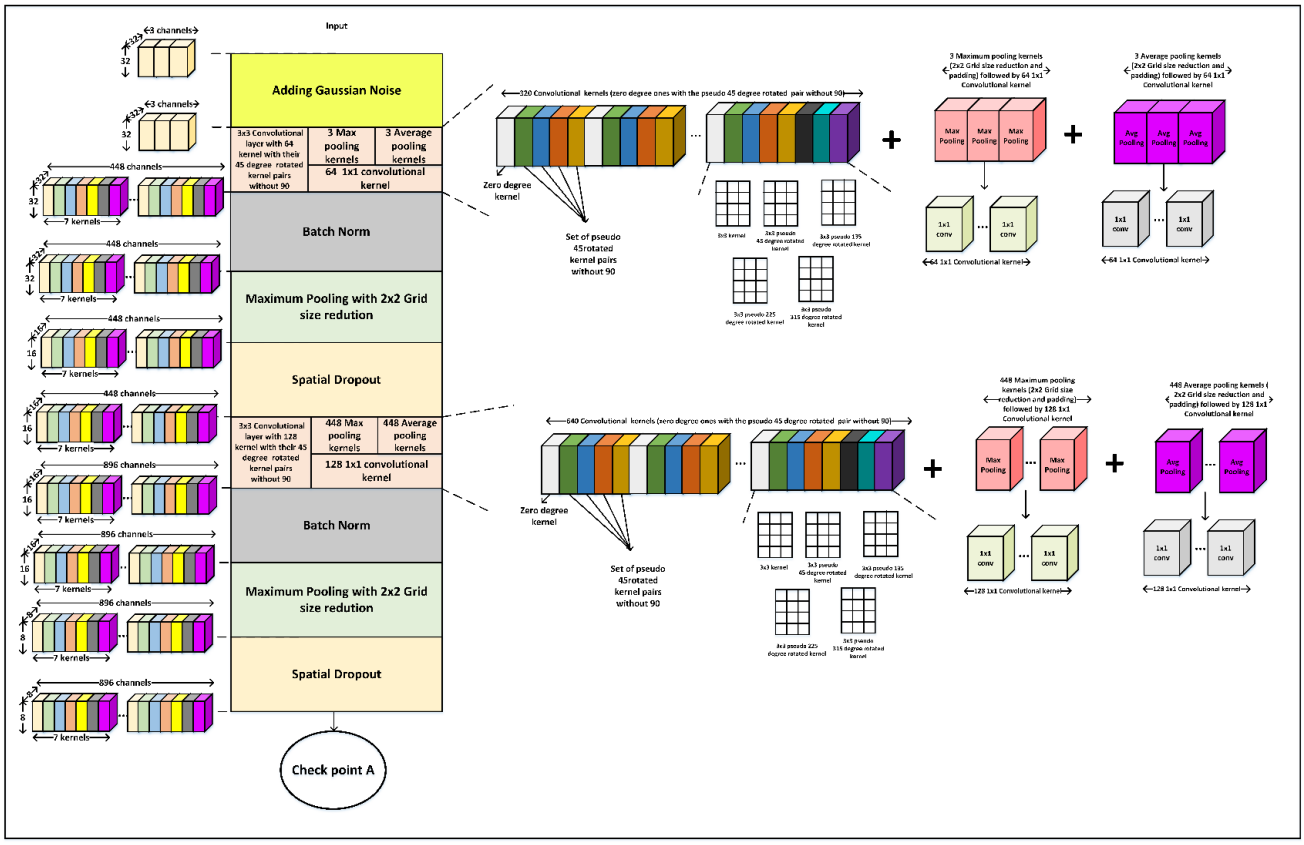

Figure 88: Pseudo Rotated VGG version 2 Part B

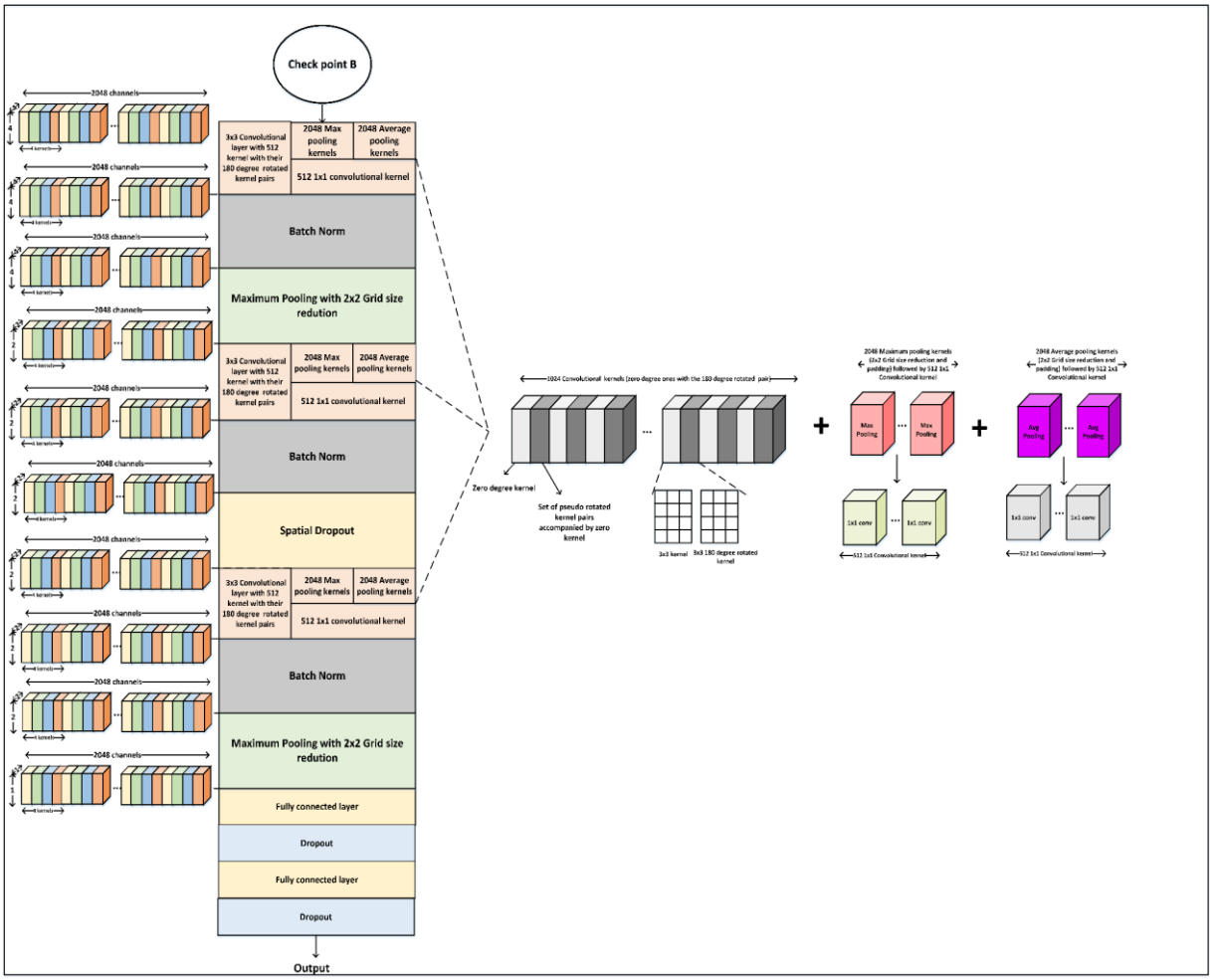

Figure 89: Pseudo Rotated VGG version 2 Part C

Chapter 6 : Performance Comparison and Benchmarking

To demonstrate how the pseudo rotated kernels contributes to the accuracy enhancements as well as its effectiveness it was required to perform three steps.

Firstly, to integrate these kernels in several networks which was done in the previous chapter through the ResNet based architectures and the VGG based ones.

Secondly, to test these architectures against several datasets which is done in this chapter through applying the ResNet based architectures on the CIFAR-100[63] while in the previous chapter both ResNet and VGG based architectures were applied on CIFAR-10[63].

Thirdly, to evaluate the achieved accuracy against previous different published image classification models.

These steps and their associated comparisons are recorded under two parts: CIFAR-10 comparison and CIFAR-100 comparison. Clearly in each part, the published networks results are listed as well as recording the rank of the proposed networks according to BenchmarksAI[110] where BenchmarksAI is a website that ranks different published networks according to their results on the given data set

6.1. CIFAR-10 Comparison

6.1.1. ResNet based Architectures

ResNet based architectures are listed in Table 14 where the results of different ResNet networks as well as their parameters ratio compared to the in house ResNet-20 are reported

Network	Top-1 error	Parameters increase ratio
In house ResNet-20	9.6 %	1x
In house ResNet-56	8.54 %	3.1x
Pseudo Rotated ResNet version 1	7.1 %	8x
Pseudo Rotated ResNet version 2	6.08 %	3.07x
Pseudo Rotated ResNet version 3	4.7 %	9.3 x
ResNet-20[29]	8.75 %	1x
ResNet-32[29]	7.51 %	1.7x
ResNet-44[29]	7.17 %	2.4x
ResNet-56[29]	6.97 %	3.1x
ResNet-110[29]	6.43%	6.2x
ResNet-1202[29]	7.93%	71.8x
ResNet-164[102]	5.46%	6.2x
ResNet-1001[102]	4.62%	37.7x

Table 14 : Comparing CIFAR-10 Pseudo Rotated ResNet versions with different ResNet available in the literature

One noteworthy to mention here, all the published ResNet networks were trained using two GPUs for large number of epochs (~ 64k epoch) with an additional number of warm up epochs (~400 epoch) unlike the proposed networks where there is a limited computational budget that limits the training to near the saturation accuracy at which the network accuracy appears to saturate without squeezing all the possible achievable accuracy from the network (almost around 700 epoch).

Moreover, these networks use multi-crop ensembles where multiple network are independently trained through different weight initialization and then are used jointly to obtain the final accuracy results on the test set.

Regardless of all of that, the results show that the Pseudo Rotated ResNet version 3 is very competitive to ResNet-1001[102] but with a significant reduction in number of parameters showing the effectiveness of increasing the width of network through the pseudo rotated kernels and pooling ones when compared to increasing the network depth through stacking more bottleneck layers.

6.1.2. VGG based Architectures

VGG based architectures are listed in Table 15 where the results of different VGG networks as well as their parameters ratio compared to the in house VGG-1 are reported

Network	Top-1 error	Parameters increase ratio
In house modified VGG-11	9.5 %	1x
Pseudo Rotated VGG version 1	7.15 %	1.9x
Pseudo Rotated VGG version 2	5.8 %	3.2x
VGG11[112]	7.91%	Not reported
VGG13[112]	6.35%	Not reported
VGG16[111]	6.75%	Not reported
VGG19[112]	6.76%	Not reported

Table 15 : Comparing CIFAR-10 Pseudo Rotated VGG versions with different VGG available in the literature

A note here, the Original VGG published paper [97] didn't experiment on the CIFAR-10 data set, so no direct results are available to compare with, however searching the literature the aforementioned papers are found.

The results show that Pseudo Rotated VGG version 2 is the one with the least error when compared to others showing how expanding the width through pseudo rotated kernels and pooling ones can generalize to different architectures.

6.1.3. Benchmarking

When benchmarking using BenchmarksAI, the CIFAR-10 data set shall have 67 different network with an accuracy ranging between 99.83% down to 75.83% where the Pseudo Rotated ResNet version 3 shall rank 17.

6.2. CIFAR-100 Comparison

6.2.1. ResNet based architectures

ResNet based architectures are listed in Table 16 where the results of different ResNet networks as well as their parameters ratio compared to the in house ResNet-20 are reported

Network	Top-1 error	Parameters increase ratio
In house ResNet-20	36 %	1x
In house ResNet-56	31.5 %	3.1x
Pseudo Rotated ResNet version 2	25.1 %	3.07x
Pseudo Rotated ResNet version 3	20.9 %	9.3 x
ResNet-164[102]	24.33%	6.2x
ResNet-1001[102]	22.7%	37.7x

Table 16 : Comparing CIFAR-100 Pseudo Rotated ResNet versions with different ResNets available in the literature

From the obtained results the Pseudo Rotated ResNet version 3 shows the least error when compared to others although the huge number of parameters difference showing the added value of the pseudo rotated kernels and the pooling ones in making the learning effective in extracting more useful features.

6.2.2. Benchmarking

When benchmarking using BenchmarksAI, the CIFAR-100 data set shall have 44 different network with an accuracy ranging between 93.51% down to 54.23% where the Pseudo Rotated ResNet version 3 shall rank 14.

Chapter 7 : Discussion and Conclusions

7.1. Summary of the work

In the era of data explosion, a huge amount of digital data is generated daily from different types of platforms such as personal computers, mobile platforms and recently the wearable devices.

Notably, images and videos are the dominant type of these data. Hence there is an urgent need for high performance computer vision tasks.

This work focused on enhancing the CNN which is considered one of the key architectures in today computer vision different tasks. CNN plays a vital role in today computer vision achievements and records from suppressing the human level accuracy in some task to the invention of a new complex applications like the autonomous vehicles. Thanks to its key features the weight sharing, feature map, channel pooling and receptive field.

In this work enhancing the CNN was done through expanding the network width by applying two main ideas the pseudo rotated kernels and attaching the pooling kernels to the convolutional layer. Both kernels allow the network to step towards a unifying more affine transformation properties within the network.

Clearly, the first type of kernels boosts the translation property through allowing the network to perform cross correlation function as well as the convolutional one; in addition to enhancing the rotation property by providing a set of arbitrary chosen rotated kernels. Meanwhile the latter promotes the scaling property.

Moreover, when combining all these kernels together the network increases its translation invariance property robustness whereas the network becomes capable to scale and rotate the feature map at each convolutional layer enriching the network capability to have its own self augmentation methods.

To demonstrate the accuracy improvement five networks were proposed based on two different architectures where three of them are based on ResNet while the remaining two are based on VGG.

Furthermore, to ensure the networks capability to generalize on different data sets, the ResNet based architectures were tested on two different data sets the CIFAR-10 and CIFAR-100.

7.2. Future work

As an extension to this work, the following points are recommended for the future work;

Firstly, to migrate all the codes to Tensor Flow version 2, this step shall require to rebuild the network from scratch to remove some obsoleted functions and classes as well as un-optimized ones. Clearly, this step shall result in a more optimized codes with less hand crafted classes which would return in a considerable reduction in the training time

Secondly, to re-explore the design circle space of pseudo rotated kernels in a more exhaustive fashion characterizing how the rotated combination affects each other and searching for other combinations that may enhance the performance more. Admittedly, the proposed combinations are just a point of kernels combination in this design circle and more performance booster combinations may exist.

Thirdly, explore the network depth dimension and study how the network can benefit from increasing both the width and depth dimensions concurrently.

Fourthly, apply some model optimization techniques such as network pruning or precision reduction. These techniques shall reduce the model size enhancing the training time as well as opening the exploration of real time applications.

Fifthly, if applicable apply the idea on a more complex data set such as the ImageNet to explore how the increasing the amount of data as well as its complexity would affect the generalization

Lastly, generalize the idea in a new application domain especially the object detection and localization one.

References

[1] M. Jordan and T. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255-260, 2015.

[2] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[3] S. Russell and P. Norvig, "Artificial intelligence: A Modern Approach," *Prentice Hall Press*, 2009

[4] V. Sze, Y. Chen, T. Yang and J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, 2017.

[5] "Keeping Our Brain Healthy," *Humanagingcentral.com*, 2020. [Online]. Available: http://www.humanagingcentral.com/brain_page.html.

[6] Z. Li, Y. Wang, T. Zhi and T. Chen, "A survey of neural network accelerators," *Frontiers of Computer Science*, vol. 11, no. 5, pp. 746-761, 2017.

[7] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[8] M. Stuart and M. Manic, "Survey of progress in deep neural networks for resource-constrained applications," *43rd Annual Conference of the IEEE Industrial Electronics Society*, Beijing, 2017.

[9] M. Shafique et al*.,* "Adaptive and Energy-Efficient Architectures for Machine Learning: Challenges, Opportunities, and Research Roadmap," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Bochum, 2017.

[10] Griffin Lacey et al "Deep Learning on FPGAs: Past, Present, and Future" *arXiv preprint arXiv*: 1602.04283, 2016.

[11] A. Ratnaparkhi, E. Pilli and R. C. Joshi, "Survey of scaling platforms for Deep Neural Networks," *International Conference on Emerging Trends in Communication Technologies (ETCT)*, Dehradun, 2016

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates,S. Bhatia, N. Boden, A. Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," *Proceedings of the 44th Annual International Symposium on Computer Architecture, ACM*, 2017.

[13] J. Dean, D. Patterson and C. Young, "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution," *IEEE Micro*, vol. 38, no. 2, pp. 21-29, 2018.

[14] M. Chen, S. Mao, Y. Zhang and V. Leung, "Big Data*"*. *Cham: Springer International Publishing*, 2014.

[15] F.-F. Li, A. Karpathy, and J. Johnson, "*Stanford CS Class CS231n: Convolutional Neural Networks for Visual Recognition".* [Online]. Available: http://cs231n.stanford.edu/

[16] "SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning? – The Official NVIDIA Blog," *Blogs.nvidia.com*, 2020. [Online]. Available: https:// blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning.

[17] B. Reagen et al., "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA),* Seoul, 2016.

[18] Eriko Nurvitadhi et al., “Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?” *Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017
[19] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1-1, 2016
[20] Soheil Hashemi et al., “Understanding the impact of precision quantization on the accuracy and energy of neural networks,” *Proceedings of the Conference on Design, Automation & Test in Europe*, Leuven, 2017.
[21] Jingyang Zhu, Zhiliang Qian and Chi-Ying Tsui, "LRADNN: High-throughput and energy-efficient Deep Neural Network accelerator using Low Rank Approximation,” *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macau, 2016.
[22] M. S. Razlighi, M. Imani, F. Koushanfar and T. Rosing, "LookNN: Neural network with no multiplication,” *Design, Automation & Test in Europe Conference & Exhibition (DATE),* Lausanne, 2017
[23] J. Zhu, J. Jiang, X. Chen and C. Tsui, "SparseNN: An energy-efficient neural network accelerator exploiting input and output sparsity,” *Design, Automation & Test in Europe Conference & Exhibition (DATE),* Dresden, 2018.
[24] Tianshi Chen et al., “DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning,” *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)* ,2014.
[25] Daofu Liu et al., “PuDianNao: A Polyvalent Machine Learning Accelerator,” *Proceedings of the 20th international conference on Architectural support for programming languages and operating systems (ASPLOS)* ,2015.
[26] S. Liu et al*.,* "Cambricon: An Instruction Set Architecture for Neural Networks," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016.
[27] Z. Du et al*.,* "Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches,” *48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.
[28] Dean, J.,” Large-Scale Deep Learning with TensorFlow for Building Intelligent Systems,” *ACM Webinar*, 2016
[29] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Las Vegas,2016
[30] Silver, D., Huang, A., Maddison, C. et al., “Mastering the game of Go with deep neural networks and tree search,” *Nature* 529, 484–489,2016.
[31] A. Ratnaparkhi, E. Pilli and R. C. Joshi, "Survey of scaling platforms for Deep Neural Networks,” *International Conference on Emerging Trends in Communication Technologies (ETCT)*, Dehradun, 2016.
[32] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing,” *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016.
[33] Ji Li et al., “Hardware-Driven Nonlinear Activation for Stochastic Computing Based Deep Convolutional Neural Networks,” *arXiv preprint arXiv*: 1703.04135, 2017.

[34] Jian Cheng et al., “Recent Advances in Efficient Computation of Deep Convolutional Neural Networks,” *arXiv preprint arXiv*: 1802.00939, 2018.
[35] Yuhao Zhu et al., “Mobile Machine Learning Hardware at ARM: A Systems-on-Chip (SoC) Perspective,” *arXiv preprint arXiv*: 1801.06274, 2018.
[36] E. Chung et al., "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave,” *IEEE Micro*,2018.
[37] B. Harris et al., "Architectures and algorithms for user customization of CNNs,” *23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju, 2018
[38] Andrew Ng, “*Deep learning specialization”.* [Online]. Available: https://www.coursera.org/specializations/deep-learning
[39] H. Jang, "Compute with Time, Not Over It: An Introduction to Spiking Neural Networks – King's Communications, Learning & Information Processing lab,” *Blogs.kcl.ac.uk*, 2020. [Online]. Available: https://blogs.kcl.ac.uk/kclip/2019/08/16/compute-with-time-not-over-it-an-introduction-to-spiking-neural-networks/.
[40] Y. Jia, et al., “Caffe: Convolutional architecture for fast feature embedding,” *Proceedings of the 22nd ACM international conference on Multimedia, MM*, 2014
[41] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie and H. Yang, "Software-Hardware Codesign for Efficient Neural Network Acceleration,” *IEEE Micro*, vol. 37, no. 2, pp. 18-25, 2017.
[42] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of monkey striate cortex,” *The Journal of Physiology*, vol. 195, no. 1, pp. 215-243, 1968.
[43] S. Han et al., "EIE: Efficient Inference Engine on Compressed Deep Neural Network,” *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, 2016
[44] H. Tann, S. Hashemi, R. I. Bahar and S. Reda, "Hardware-software codesign of accurate, multiplier-free Deep Neural Networks,” *54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, 2017.
[45] Y. Shen, M. Ferdman and P. Milder, "Escher: A CNN Accelerator with Flexible Buffering to Minimize Off-Chip Transfer,” *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, 2017
[46] Z. Li, A. Ren, J. Li, Q. Qiu, Y. Wang and B. Yuan, "DSCNN: Hardware-oriented optimization for Stochastic Computing based Deep Convolutional Neural Networks,” *IEEE 34th International Conference on Computer Design (ICCD)*, Scottsdale, AZ, 2016
[47] K. Guo *et al*., "Angel-Eye: A Complete Design Flow for Mapping CNN On to Embedded FPGA,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35-47, 2018.
[48] Y. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127-138, 2017
[49] L. Lu, Y. Liang, Q. Xiao and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs,” *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, 2017
[50] L. Du et al., "A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 198-208, 2018
[51] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong,” Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural

Networks," *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA),* 2015
[52] Kalin Ovtcharov et al., "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," *Microsoft Research*, 2015.
[53] Franyell Silfa, Gem Dot, Jose-Maria Arnau, and Antonio Gonzàlez, "E-PUR: an energy-efficient processing unit for recurrent neural networks," *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques (PACT*), 2018
[54] K. Mohamed, "*Neuromorphic Computing and Beyond*," New York: Springer, 2020.
[55] M. M. Khan et al., "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor," *IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence),* 2008.
[56] N. Srinivasa and J. M. Cruz-Albrecht, "Neuromorphic adaptive plastic scalable electronics: analog learning systems," *IEEE Pulse*, vol. 3, no. 1, pp. 51–56, 2012.
[57] P. A. Merolla et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
[58] Complete Visual Networking Index (VNI) Forecast, *Cisco*, San Jose, CA, USA,2016
[59] J. Woodhouse, "Big, Big, Big Data: Higher and Higher Resolution Video Surveillance," [Online]. Available: http://technology.ihs.com
[60] Y. LeCun, et al., "Handwritten digit recognition: Applications of neural network chips and automatic learning," *IEEE Communication Magazine*, vol. 27, no. 11, pp. 41–46, 1989
[61] C. Szegedy et al., "Going deeper with convolutions," *Proc. CVPR*, 2015
[62] C. J. B. Yann, Y. LeCun, and C. Cortes, "The MNIST DATABASE of Handwritten Digits". [Online]. Available: http://yann.lecun.com/exdb/mnist/
[63] Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 Dataset". [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html
[64] O. Russakovsky et al*.,* "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
[65] Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Proceedings of NIPS*, 2012.
[66] Pascal VOC Data Sets. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/
[67] Microsoft Common Objects in Context (*COCO)* Dataset. [Online]. Available: http://mscoco.org/
[68] Google Open Images. [Online]. Available: https://github.com/openimages/dataset
[69] YouTube-8M. [Online]. Available: https://research.google.com/youtube8m/
[70] AudioSet. [Online]. Available: https://research.google.com/audioset/index.html
[71] Hinton, G. et al, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Mag*azine, 2012
[72] R. Yazdani, A. Segura, J. Arnau and A. Gonzalez, "An ultra-low-power hardware accelerator for automatic speech recognition," *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO),* Taipei, 2016
[73] C. Lopes and F. Perdigao, "Phone recognition on the TIMIT database," *Speech Technologies/Book*, vol. 1, pp. 285-302, 2011.
[74] Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," *arXiv preprint* arXiv:1706.08612, 2017.

[75]] J. Barker, S. Watanabe, E. Vincent, and J. Trmal, "The fifth'CHiME'Speech Separation and Recognition Challenge: Dataset, task and baselines," *arXiv preprint* arXiv:1803.10609, 2018.
[76] T. Afouras, J. S. Chung, and A. Zisserman, "LRS3-TED: a large-scale dataset for visual speech recognition," *arXiv preprint* arXiv:1809.00496, 2018
[77] Z. Zhao, P. Zheng, S. Xu and X. Wu, "Object Detection with Deep Learning: A Review," IEEE *Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212-3232, 2019
[78] Esteva, et al*.,* "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.
[79] R. Girshick, "Fast r-cnn," *IEEE International Conference on Computer Vision (ICCV),* Santiago, 2015
[80] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016
[81] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 2017
[82] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, 2014
[83] F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston2015
[84] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," *IEEE Conference on Computer Vision and Pattern Recognition*, 2014
[85] "OpenFace," *Cmusatyalab.github.io*, 2020. [Online]. Available: https://cmusatyalab.github.io/openface/
[86] Rodríguez-Moreno, Itsaso, José María Martínez-Otzeta, Basilio Sierra, Igor Rodriguez, and Ekaitz Jauregi, "Video activity recognition: State-of-the-art," *Sensors* ,2019
[87] L. Wang et al, "Towards good practices for very deep two-stream convNets," *arXiv preprint* arXiv:1507.02159,2015
[88] Ullah, J. Ahmad, K. Muhammad, M. Sajjad and S. W. Baik, "Action Recognition in Video Sequences using Deep Bi-Directional LSTM with CNN Feature," *IEEE Access*, 2018
[89] X. Wang, L. Gao, P. Wang, X. Sun and X. Liu, "Two-Stream 3-D convNet Fusion for Action Recognition in Videos with Arbitrary Size and Length," *IEEE Transactions on Multimedia*, vol. 20, no. 3, pp. 634-644, 2018
[90] E. Holliman, J. Godfrey and J. McDaniel, "SWITCHBOARD: telephone speech corpus for research and development," *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, San Francisco,1992
[91] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang and A. Stolcke, "The Microsoft 2017 Conversational Speech Recognition System," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Calgary, 2018
[92] Graves, A. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks*," IEEE International Conference on Acoustics, Speech and Signal Processing*, Vancouver, 2013

[93] W. Chan, N. Jaitly, Q. Le and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, 2016
[94] Y. Zhang, W. Chan and N. Jaitly, "Very deep convolutional networks for end-to-end speech recognition," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, New Orleans, LA, 2017
[95] T. N. Sainath, O. Vinyals, A. Senior and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks*," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Brisbane, QLD, 2015
[96] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated recognition, localization and detection using convolutional networks," *Proceeding of ICLR*, 2014
[97] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceeding of ICLR*, 2015
[98] C. Szegedy et al., "Going deeper with convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Boston,2015
[99] S. Ioffe and C. Szegedy," Batch normalization: accelerating deep network training by reducing internal covariate shift," *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML)*, 2015
[100] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* Las Vegas, 2016
[101] C. Szegedy et al., "Inception-v4, inception-ResNet and the impact of residual connections on learning," *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI),*2017
[102] K. He, X. Zhang, S. Ren and J.Sun, " Identity Mappings in Deep Residual Networks," *arXiv preprint arXiv*: 1603.05027, 2016.
[103] Y. Cheng, D. Wang, P. Zhou, T. Zhang," A Survey of Model Compression and Acceleration for Deep Neural Networks," *arXiv preprint* arXiv:1710.09282,2017
[104] Matthew D Zeiler, Rob Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv preprint arXiv*: 1311.2901, 2013.
[105] M. Lin, Q. Chen, and S. Yan, "Network in network," *Proceeding ICLR*, 2014.
[106] Canziani, A. Paszke, E. Culurciello," An analysis of deep neural network models for practical applications," *arXiv preprint* arXiv:1605.07678
[107] K. Team, "Keras: The Python deep learning API," Keras.io, 2020. [Online]. Available: https://keras.io/
[108] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machining Learning Research*, 2014
[109] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun and Christoph Bregler, "Efficient Object Localization Using Convolutional Networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015
[110] "Benchmarks.AI - Directory of AI Benchmarks," *Benchmarks.ai*, 2020. [Online].Available: https://benchmarks.ai
[111] Hao Li et al., "Pruning Filters for Efficient ConvNets," *Proceedings of the 5th International Conference on Learning Representations (ICLR),* 2017

[112] B. O. Ayinde, T. Inanc and J. M. Zurada, "On Correlation of Features Extracted by Deep Neural Networks," *International Joint Conference on Neural Networks (IJCNN),* Budapest, 2019.
[113] V. Mnih, et al., "Playing atari with deep reinforcement learning," *Proceedings of NIPS Deep Learning Workshop*, 2013
[114] Urs Köster et al., "Flexpoint: an adaptive numerical format for efficient training of deep neural networks," *In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS*), 2017
[115] Philipp Gysel, Mohammad Motamedi and Soheil Ghiasi," Hardware-oriented Approximation of Convolutional Neural Networks," *arXiv preprint* arXiv: 1604.03168,2016
[116] Song Han et al., "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," *In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)* 2017.
[117] Y. Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," *Energy* ,2018
[118] Parker Hill, et al. "Rethinking numerical representations for deep neural networks," *arXiv* preprint arXiv:1808.02513,2018
[119] Tim Dettmers, "8-bit approximations for parallelism in deep learning," *arXiv* preprint arXiv:1511.04561 ,2015
[120] D. Shin, J. Lee, J. Lee and H. Yoo, "14.2 DNPU: An 8.1TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks," *IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, 2017
[121] P. N. Whatmough, S. K. Lee, H. Lee, S. Rama, D. Brooks and G. Wei, "14.3 A 28nm SoC with a 1.2GHz 568nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications," *IEEE International Solid-State Circuits Conference (ISSCC),* San Francisco, CA, 2017
[122] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *In Proceedings of MICRO*, 2016
[123] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," *In Proceedings of NIPS*, 2015
[124] M. Courbariaux and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," *arXiv* preprint arXiv:1602.02830,2016
[125] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," *In Proceedings of ECCV*, 2016
[126] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by halfwave Gaussian quantization," *In Proceedings of CVPR*, 2017.
[127] F. Li and B. Liu, "Ternary weight networks," *In Proceedings of NIPS Workshop Efficient Methods Deep Neural Network*, 2016.
[128] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *In Proceedings of ICLR,* 2017.
[129] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logrithmic computations," *In Proceedings of ICASSP*, 2017
[130] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," *In Proceedings of ICLR*, 2017.

[131] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *In Proceedings of ICLR*, 2016.
[132] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” *In Proceedings of NIPS,*1990.
[133] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *In Proceedings of NIPS*, 2015
[134] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *In Proceedings of NIPS Deep Learn*. Workshop, 2014.
[135] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through FFTs,” *In Proceedings of ICLR*, 2014.
[136] A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” *In Proceedings of CVPR*, 2016.
[137] J. Cong and B. Xiao, “Minimizing computation in convolutional neural networks,” *In Proceedings of ICANN*, 2014.
[138] V. Lebedev, Y. Ganin, M. Rakhuba1, I. Oseledets, and V. Lempitsky, “Speeding up convolutional neural networks using fine-tuned CP-decomposition,” *In Proceedings of ICLR*, 2015
[139] Hardik Sharma, et al. "Dnnweaver: From high-level deep network models to fpga acceleration," *In the Workshop on Cognitive Architectures*, 2016.

Appendix A: Keras Flow

Keras is the framework used across the implementation of all networks within this work. It can be described as an open source python based deep neural network library whereas a multi-level hierarchy of libraries are constructed. The structure has a higher level neural network API libraries built on the top of a backend lower level libraries such as Tensor Flow, Theano or Microsoft CNTK which are capable to execute seamlessly on both CPU and GPU as shown in Figure 90 . Keras flavor used among this work is the one with tensor flow as backend.

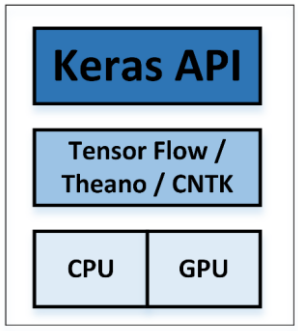

Figure 90: Keras levels structure

It distinct features are the ability to enable fast prototyping through autonomously handling the common infrastructure details such as the back propagation algorithm and the optimization procedures as well as benefiting from being open source through a wide online community support.

Keras flow can be divided into five major steps as shown in Figure 91

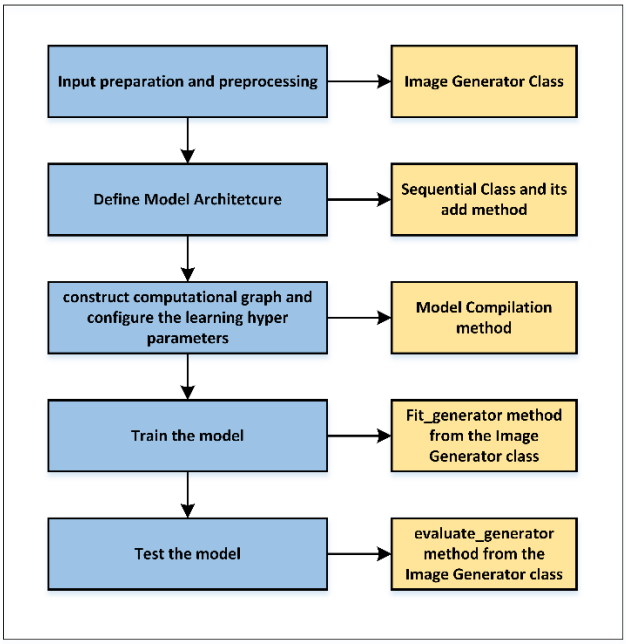

Figure 91: Keras flow

Firstly, prepare the inputs, perform any preprocessing required and pass it to the network. This is done through the Image Data generator class. This class solves the hassle of how to feed the network with the data whether to load them in the RAM or fetch them every time from the hard disk or write a manual code that can handle the data movement across the different available storage hierarchy within the computing platform based on the size of data. This class handles all the data set loading automatically whereas the images are divided into batches and only images that are required for the current and next few batches during training are loaded in the memory. Clearly, this shall allow loading both small datasets as well as very large image datasets with thousands or millions of images smoothly in and out from memory. This can be noted as progressive loading, as the data set is progressively loaded and retrieving just enough data for what is needed immediately. Moreover, Image Data generator class can be used in image augmentation to improve the networks performance whereas it can automatically scale the pixel values of the images as well as automatically create transformed versions of images that belong to the same class as the original image. These Transforms include a range of operations from the field of image manipulation, such as shifts, flips, zooms, rotate and may other operations. To use the Image Data Generator class, the data set directory shall be structure as shown in Figure 92

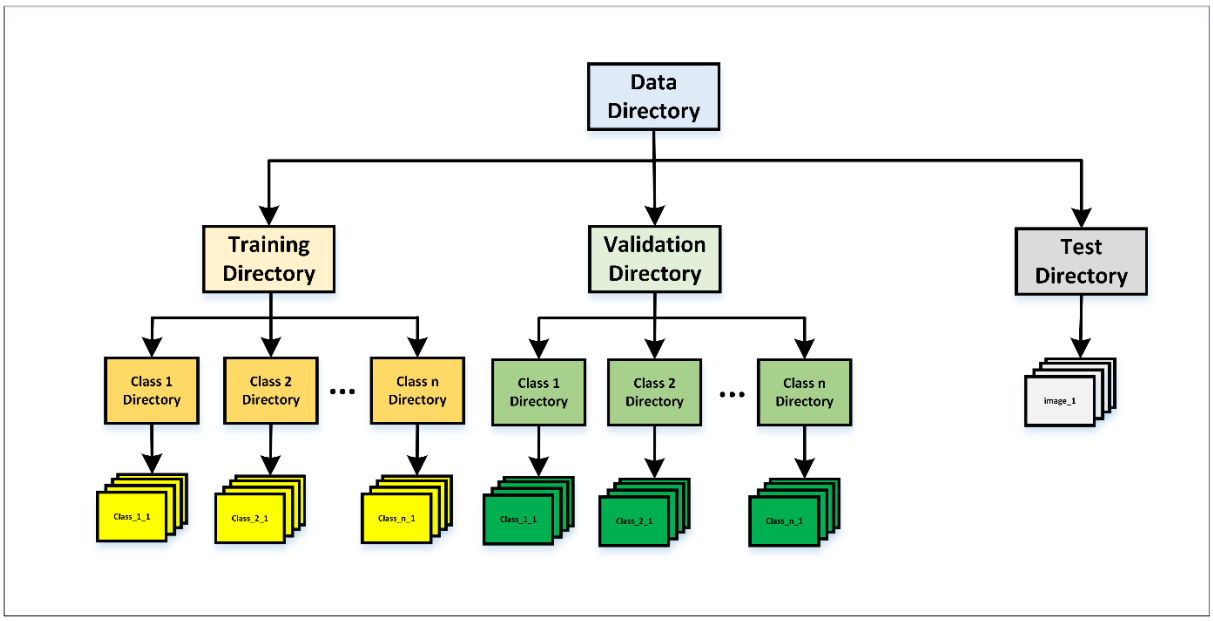

Figure 92 : keras Image data generator class directory structure

This shall be followed by creating a constructor for this class through “datagen = ImageDataGenerator()”. Then create the training generator " train_datagen = ImageDataGenerator (various required options)” where the options include rescaling the data and any required image augmentation configurations such as shift, flip, zoom, brightness and rotation. Similarly, a test generator is created via “test_datagen = ImageDataGenerator(). Finally, instantiating different iterators to progressively load the data. This is done by calling the “flow_from_directory” function. For training generator it is called as “train_generator = train_datagen.flow_from_directory(path to train directory, batch size, shuffling data)” while for validation as “validation_generator = train_datagen.flow_from_directory(path to validation directory, batch size, shuffling data)” and for testing as “test_generator = test_datagen.flow_from_directory(path to test directory, batch size, shuffling data)”.

Secondly, define the model architecture. This can be done through sequential class from keras models via “model = Sequential ()”. Then different layers are stacked sequentially through the add method as “model.add(layer name(layer configuration))”.

Thirdly, construct the computational graph and configure the learning hyper parameters by compiling the model. This is done through “model.compile(loss= target cost function, ,optimizer=required optimization method , metrics= required merit of optimization)”

Fourthly, start training the model via “model.fit_generator(train_generator, steps_per_epoch, epochs, validation, validation_steps” where training and validation steps are the number of batches per epoch

Lastly, test the model on test data through “model.evaluate_generator (test_generator, steps)”

Appendix B: Computing Platforms

DNN are widely known to be computational hungry given their huge size, number of computations required for their training and the amount of associated memory to store the model as well as the intermediate results. Nowadays, training is usually done through one or several high end GPUs with a huge RAM memory size. This kind of infrastructure are commonly found in data centers with an emphasis on Google Colaboratory (open GPUs from google), Google compute engine, Amazon EC2 and Microsoft Azure. Data centers strength lies in their reduced cost of ownership as well as offering a more data computation centric GPUs compared to building a customized platform for a short term usage.

Google Colaboratory is a google based service that provides a Jupyter notebook environment that doesn't require any setup and runs entirely on google cloud. It is equipped with two different hardware accelerators where only one is allowed to be used at a time. The first is Nvidia K80 GPU with 12 GB of RAM while the other is google Tensor Processing Unit (TPU). This service comes for free, however only a maximum continues run of twelve hours is allowed before resetting the connection as well as the quality of service and allocating a hardware accelerator isn't guaranteed.

Google Compute Engine is the google paid service where a virtual machine connected to google data center is offered. This virtual machine can be equipped with Xeon processor with different number of cores and amount of RAM. Moreover, it is allowed to attach a GPU to this machine for a wide range of GPUS including Nvidia V100, P100, K80, P4 and T4

Amazon EC2 is an amazon paid service that similarly to google compute engine can provide a virtual machine connected to amazon data centers that is equipped with Xeon processor with different number of cores and amount of RAM. Also, the computation capability can be extended through attaching one or multiple GPUs from the available ones including Nvidia V100, K80 and M60

Microsoft Azure is the microsoft paid service that similarly to the others provides a virtual machine equipped with Xeon processor with an option to choose the number of cores, amount of RAM and attaching a GPU from the available ones including Nvidia V100, P100, P40, k80 and M60

Table 17 shows a comparison between different attached GPUs of these platforms as well as their pricing while Table 18 shows when the key advantage of each platform and when to use it

Platform	Available CPU	Available Nvidia GPU	Price per Hour in $
Google Colaboratory	N/A	K80	0
Google Compute Engine	Intel Xeon	K80	0.7
Google Compute Engine	Intel Xeon	P4	0.9
Google Compute Engine	Intel Xeon	T4	1.24
Google Compute Engine	Intel Xeon	P100	3

Google Compute Engine	Intel Xeon	V100	4.5
Amazon EC2	Intel Xeon	K80	0.9
Amazon EC2	Intel Xeon	M60	0.93
Amazon EC2	Intel Xeon	V100	3.06
Microsoft Azure	Intel Xeon	K80	0.9
Microsoft Azure	Intel Xeon	M60	1.092
Microsoft Azure	Intel Xeon	P40	2.07
Microsoft Azure	Intel Xeon	P100	2.07
Microsoft Azure	Intel Xeon	V100	3.06

Table 17 : Different platforms computing capability and their pricing

Platform	Key Advantage	When to use
Google Colaboratory	Totally free service	Developing and experimenting small functions
Google Compute Engine	The 300$ voucher which is equivalent to training one month for free Preemptive machines which are lower in price where a V100 can be as low as 1$, however it lasts only from 2 up to 12 hours	Prototyping a full network and network debugging
Amazon EC2	Least V100 GPU price Spot machines which is lower are price where a V100 can be as low as 2$, however it was very difficult to found one	Training a full network that may require running for several days
Microsoft Azure	Competitive V100 GPU price	Training a full network that may require running for several days

Table 18 : key advantage of each platform and when to be used

الملخص

في عصر انفجار البيانات ، اصبح استخدام مصطلحات مثل "البيانات الضخمه" شائعاً حيث تم ربط العالم ورقمنته من خلال التوفر الواسع لمنصات الحوسبة الشخصية المتصلة بالشبكة العنكبوتية ، الانتشار السريع لمنصات الهاتف المحمول ، شعبية تطبيقات الوسائط الاجتماعية وبدء عصر إنترنت الأشياء مصحوبًا باختراع أجهزة ذكية تُستخدم تقريبًا في جميع جوانب الحياة من الأجهزة القابلة للارتداء إلى أجهزة المنزل المختلفه. كل ما سبق ، نتج عنه توليد يومي لكمية هائلة من البيانات الرقمية مثل المستندات و الصور و المقاطع الصوتية و المرئية. يتميز هذا النوع من البيانات بطابعه الشخصي مما ادى الى جاذبية لاستخدام أساليب تعلم الآله لاستخراج رؤى وتنبؤات ومعلومات مفيدة منها. علاوة على ذلك ، فإن حوالي 70٪ من هذه البيانات عبارة عن صور ومقاطع مرئية مما يزيد من متطلبات تحسين تطبيقات الرؤية الحاسوبية. الشبكة العصبونية التلافيفية التي تعد مجالًا فرعيًا من تعلم الآله كانت اللاعب الرئيسي في تعزيز و تحسين تطبيقات الرؤية الحاسوبية المستخدمة اليوم بفضل خصائصها المميزة مثل مشاركة مصفوفة الاوزان وخريطة السمات والمجال الاستقبالي و انتقاء القنوات المختلفة و تجميعها.

يستكشف هذا العمل تعزيز أداء الشبكة العصبونية التلافيفية عن طريق زياده البعد العرضى لها. يتم ذلك من خلال فكرتين رئيسيتين. أولاً تدوير المرشحات التلافيفية باستدارة مستعارة حيث يتم تدوير المرشحات المدربة بزوايا دوران مستعارة مختلفة لتوليد متغيرات متعددة منها. ثانيًا ، إرفاق المرشحات التجميعية بالطبقة التلافيفية. كل هذا سمح للشبكة أن تتقدم أكثر نحو توحيد العديد من خصائص التحويل الأفيني بداخلها. بشكل اكتر وضوحاً، يتم تعزيز خاصية الانعكاس والدوران من خلال توفير مجموعة من المرشحات ذات الاستدارة المستعارة المختارة بشكل اعتباطى بينما يعزز خاصية التحجيم من خلال التغيير الاختيارى لحجم خرائط السمات.علاوة على ذلك ، فإن كل هذه المرشحات المجتمعة توفر للشبكة القدرة على زيادة و تنويع خرائط السمات بداخل كل طبقة تلافيفية مما يزيد من متانة خاصية ثبات الترجمة. لإثبات تحسن الأداء ، تم اقتراح خمس شبكات تعتمد على بنيتين مختلفتين بالإضافة إلى التاكد من أدائهما من خلال اختبارهما على مجموعتين مختلفتين من البيانات

مهندس: محسن رأفت عبدالعاطى سيد

تاريخ الميلاد: ١٩٩٢\٠٢\٢٢

الجنسية: مصرى

تاريخ التسجيل: ٢٠١٥\١٠\١١

تاريخ المنح: ٢٠٢٢

القسم: هندسة الإلكترونيات والإتصالات الكهربية

الدرجة: ماجستير العلوم

المشرفون:

ا.د. محسن عبد الرازق رشوان

ا.د. حسام على حسن فهمى

الممتحنون:

أ.د. محسن عبد الرازق رشوان (المشرف الرئيسي)

أ.د. حسام على حسن فهمى (المشرف)

أ.م.د.عمر نصر (الممتحن الداخلي)

أ.د خالد مصطفى (الممتحن الخارجي)

أستاذ بكلية الحاسبات والذكاء الاصطناعي – جامعة القاهرة

عنوان الرسالة:

شبكات الاستدارة المستعارة : توسعه الشبكة العصبونية التلافيفية بواسطة المرشحات ذات الاستدارة المستعارة

الكلمات الدالة: (يجب أن تتكون من 5 كلمات فقط)

تعلم الآلة ، الشبكة العصبونية التلافيفية ، تصنيف الصورة ، مرشحات الاستدارة المستعارة ، مرشحات التجميع.

ملخص الرسالة: (لا يزيد عن 150 كلمة ولا يتخطى صفحة أخري)

يهدف هذا العمل إلى تعزيز أداء الشبكة العصبونية التلافيفية من خلال استكشاف زيادة البعد العرضى.الفكرة المقترحة هي تدوير المرشحات التلافيفية باستدارة مستعارة لإنشاء نسخ متعددة من تلك المدربة في الأصل حيث يتم تدوير كل مرشح بزاوية دوران مختلفة. علاوة على ذلك ، فإن ادماج هذه المرشحات ذات الاستداره المستعارة مع المرشحات المجمعة من شأنه أن يجعل الشبكة تتقدم أكثر نحو توحيد العديد من خصائص التحويل الأفيني بداخلها. يمكن أيضًا رؤية ذلك ايضا كما لو أن الشبكة اصبحت قادرة على زيادة و تنويع خرائط السمات بداخلها. لإثبات فعالية هذه الأفكار ، تم اقتراح خمس شبكات تعتمد على بنيتين مختلفتين وتم تعميمهما ليتم اختبارهما على مجموعتين مختلفتين من البيانات

شبكات الاستدارة المستعارة : توسعة الشبكة العصبونية التلافيفية بواسطة المرشحات ذات الاستدارة المستعارة

اعداد

محسن رأفت عبدالعاطى سيد

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الإلكترونيات والإتصالات الكهربية

يعتمد من لجنة الممتحنين:

أ.د. محسن عبد الرازق رشوان	**المشرف الرئيسى**
أ.د. حسام على حسن فهمى	**مشرف**
أ.م.د. عمر نصر	**الممتحن الداخلي**
أ.د. خالد مصطفى	**الممتحن الخارجي**

- أستاذ بكلية الحاسبات والذكاء الاصطناعي - جامعة القاهرة

كلية الهندسة - جامعة القاهرة

الجيزة - جمهورية مصر العربية

٢٠٢٢

شبكات الاستدارة المستعارة : توسعة الشبكة العصبونية التلافيفية بواسطة المرشحات ذات الاستدارة المستعارة

اعداد

محسن رأفت عبدالعاطى سيد

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة

كجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الإلكترونيات والإتصالات الكهربية

تحت اشراف

أ.د. محسن عبد الرازق رشوان .....

أستاذ

قسم هندسة الإلكترونيات والإتصالات الكهربية

كلية الهندسة - جامعة القاهرة

أ.د. حسام على حسن فهمى .....

أستاذ

قسم هندسة الإلكترونيات والإتصالات الكهربية

كلية الهندسة - جامعة القاهرة

كليــة الهندســة - جامعــة القاهــرة

الجيـزة - جمهوريـة مصـر العربيــة

٢٠٢٢

شبكات الاستدارة المستعارة : توسعة الشبكة العصبونية التلافيفية بواسطة المرشحات ذات الاستدارة المستعارة

اعداد

محسن رأفت عبدالعاطى سيد

رسالة مقدمة إلى كلية الهندسة – جامعة القاهرة
كجزء من متطلبات الحصول على درجة
ماجستير العلوم
في
هندسة الإلكترونيات والإتصالات الكهربية

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة

٢٠٢٢