Convolutional Neural Networks Progress: Architectural and Optimization Methods Survey

Mohsen Raafat*1, Hossam A.H.Fahmy*2, Mohsen Rashwan*3

*Electronics and Communication Department, Faculty of Engineering , Cairo University
Giza, 12613, Egypt

[1]moh_raafat@hotmail.com

[2]hossam.a.h.fahmy@gmail.com

[3]mrashwan@rdi-eg.ai

Abstract: ***Since the start of the Convolutional Neural Networks (CNN) paradigm, they were applied in a wide range of computer vision tasks such as image classification, object detection, localization, tracking and action recognition where they were able to show breakthrough performance and generate a new state of the art results. This paper surveys the progress of CNN from an architectural and optimization perspectives. While many CNN reviews exist in the literature, most of them had focused on providing a survey either from a network architecture prospective or an application one, unlike this one which provides a brief general overview for the key features of CNN, followed by reviewing the progress of the state of the art architectures and finally considers the change in the merit of figure of how the CNN are evaluated to include the optimization methods to provide practical CNN that can be deployed on today's hardware infrastructure without significantly impacting the achieved accuracy.***

Key words: ***machine learning (ML); deep neural networks (DNN), convolutional neural networks (CNN).***

1 INTRODUCTION

Convolutional Neural Networks (CNN) had progressed starting from LeNet-5[1] which was designed for simple grayscale digit recognition until the ResNet[2] that were applied on ImageNet competition[3] with around 1.2 million color resolution achieving an accuracy 3.5% suppressing the human level performance. Undoubtedly, their distinct features as well as their superior performance that outperformed the previous state of the art approaches had enabled their usage in a wide range of applications.

Applications that use CNN Starts from image classification whereas the aforementioned ResNet[2] was able to suppress the human level performance. Moving to object detection and localization where the invention of Regions within CNN framework (R-CNN) [4] had achieved a significant performance improvement allowing real time accurate object detection applications. The region based CNN unified framework paradigm had continued to improve through set of proposals including Fast R-CNN [5], Faster R-CNN [6] and YOLO [7]. Needless to mention, face recognition applications that had benefited significantly from the CNN superior performance through the proposed face recognition state of the art networks such as Google's FaceNet [8] which is based on training the CNN with a triplet loss function to allow the network to learn to cluster the face representations of the same person, Facebook's DeepFace [9] 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 [10] an open-source face recognition tool. Moreover, action and activity recognition application which is one of the most challenging tasks that involves identifying human activities from an image or video sequences had been improved through the progress of CNN achieving a new state of the art results but still away from the level of impact brought to image classification. The current state of the art is a dual architecture that combines both CNN and LSTM [11,12,13]. Furthermore, CNN had started to be used in more critical applications such as medical analysis where they are used to detect whether a disease exists or not. For instance, they are used in diagnosis of different kinds of cancers from brain [14] to skin [15] and breast [7] with an achieved competitive performance to the human proficient.

This paper aims to introduce the CNN key features that enabled its distinct performance, then it surveys the state of the art networks and finally shows the proposed optimization methods to reduce its computational complexity and the network size.

2 CNN OVERVIEW

CNN are mainly considered as an extension from DNN that is capable to operate on data that has a temporal or spatial continuity nature. They were inspired from [16] 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 have two dimensional format and videos have three dimensional format.

A. 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.

1) Receptive Field

Receptive field as shown in Figure 1 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.

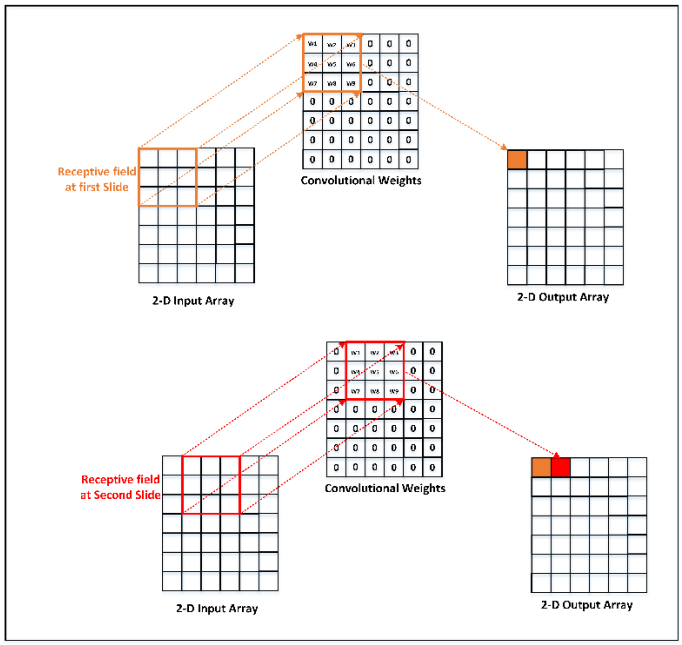

Figure 1 : Receptive field for two sliding windows

2) Feature map

Feature map which is shown in Figure 2 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.

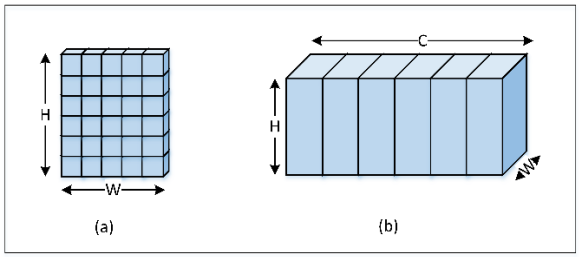

Figure 2 : (a) Feature map with single channel (b) Feature map with C channels

3) Channel pooling

Channel pooling which can be viewed in Figure 3 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.

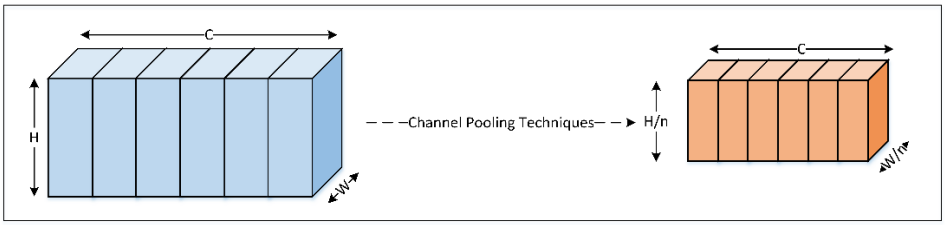

Figure 3 : Feature map before and after channel pooling where n is the pooling scaling value

4) Shared Weights

Feature map which is shown in Figure 4 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.

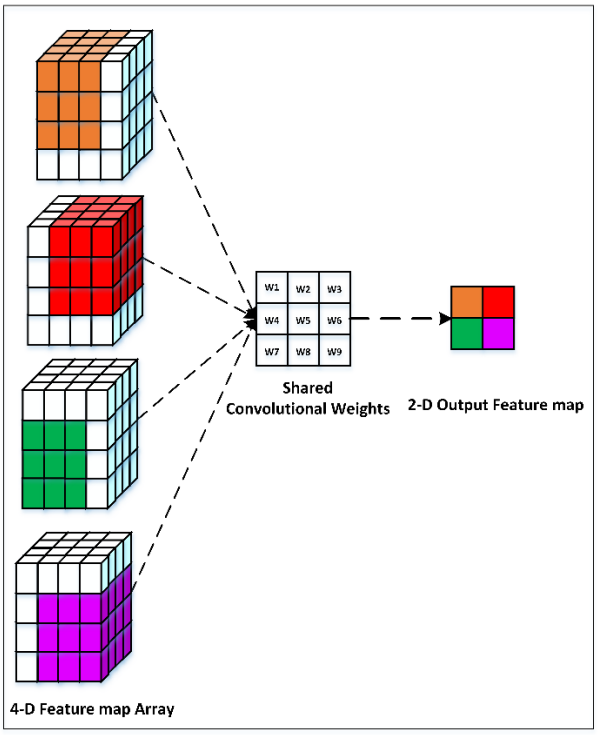

Figure 4 : Feature map with four channels where the same kernel is applied across the entire map to generate an output feature map with single channel

B. Typical CNN Architecture

A typical CNN architecture as shown in Figure 5 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.

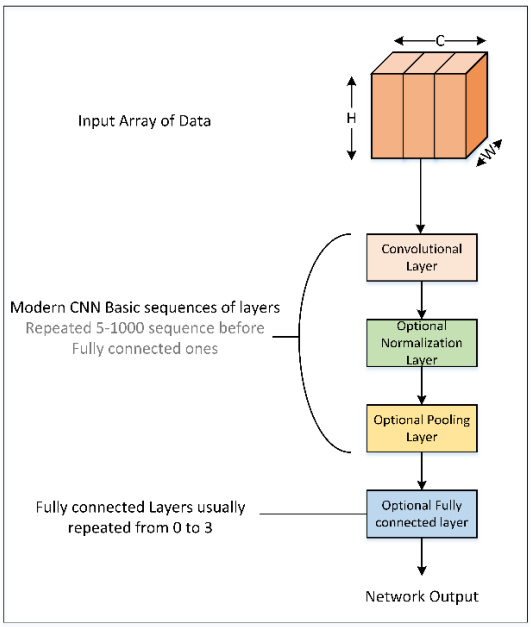

Figure 5 : Typical Modern CNN different layer structure

1) Convolutional layer

Convolutional Layer gets its name from the fact that their operation is a convolution one. This is the layer where the dominant number of computations of the CNN takes place. As shown in Figure 6, 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 from 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.

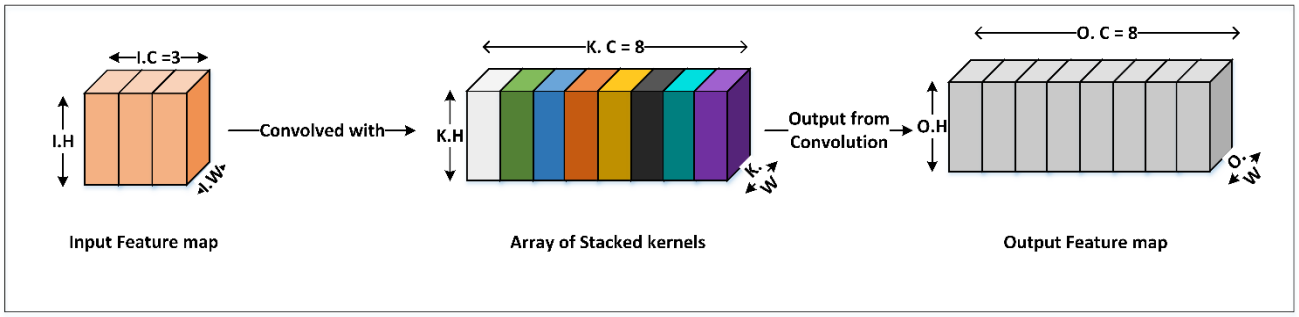

Figure 6 : Example for the convolutional layer where an input feature map with 3 of channels

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 7 where the stride defines the step window of the non-overlapping blocks associated with separate example for both maximum and average pooling.

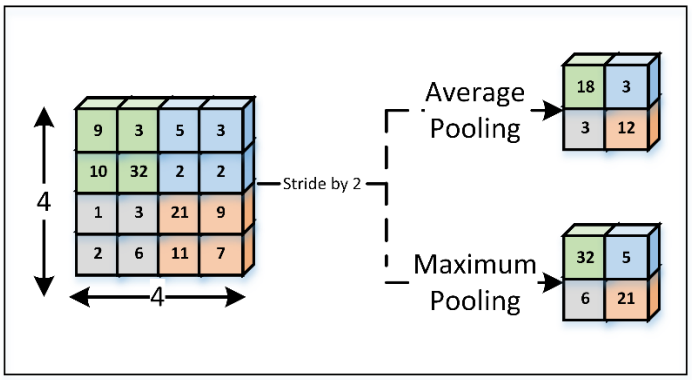

Figure 7 : Feature map with a single channel is reduced through average and maximum pooling with striding by 2

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 output logits (the softmax function output scores) from 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.

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 iterations. Also, it enhances the achieved accuracy by introducing some noise in the data allowing a better generalization. There are many types of these layers 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

3 CNN Architecture Progress

During the last several decades, many CNN architectures have 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 [3] 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[17] was introduced by a group from Toronto university where 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 8, the CNN had continued to improve the error rate in the ImageNet challenge with a significant milestone at 2015 when the ResNet[2] had been introduced whereas it was able to surpass the human level accuracy. Furthermore, from [3] 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. In this section, the progress of the CNN is explored starting from LeNet-5 until the ResNet

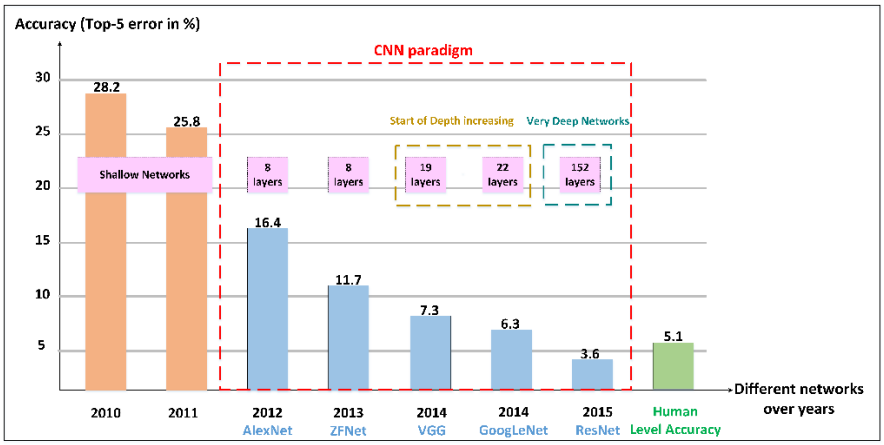

Figure 8 : ImageNet top-5 error accuracy versus different networks progress over years inspired from [3]

A. LeNet-5

LeNet-5[1] was introduced in 1989 as one of the first CNN that was designed for the digit classification task on the MNIST grayscale images [18]. Handwritten 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 9, 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.

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

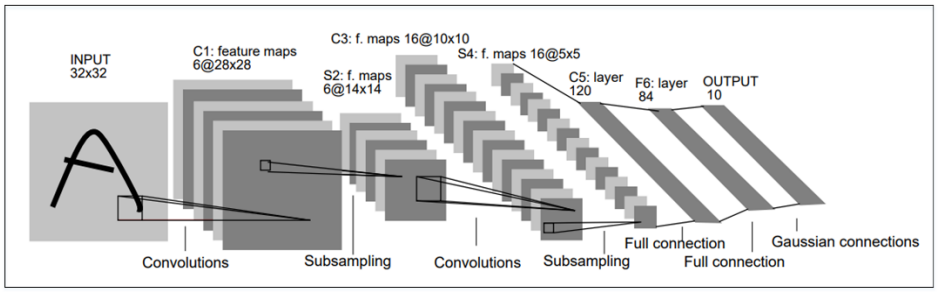

Figure 9 : LeNet-5 architecture from [1]

B. AlexNet

AlexNet [17] was introduced in 2012 and the first CNN based network to win the image classification track within the ImageNet Challenge.

As shown in Figure 10, it is composed of five convolutional layers, three maximum pooling layers 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.

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.

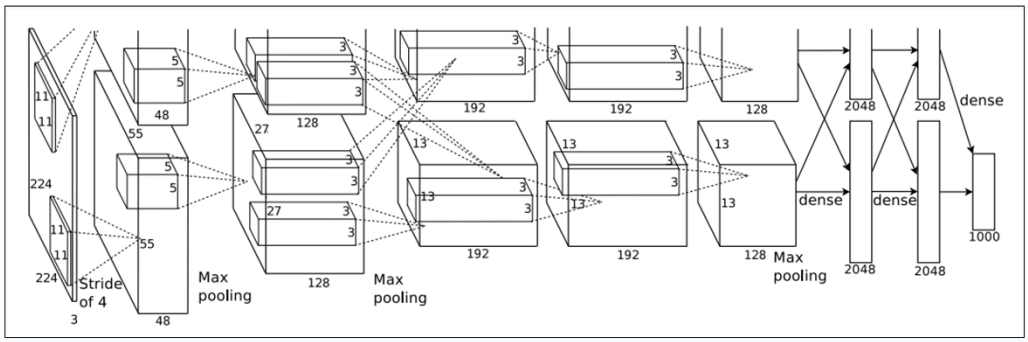

Figure 10 : AlexNet architecture from [17]

C. ZFNet

ZFNet[19] 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.

D. Overfeat

Overfeat [20] 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 the number of kernels is 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. Recommended font sizes are shown in Table 1.

E. VGG

VGG [21] 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 11 where a 5x5 kernel can have the same effective receptive field of two stacked 3x3 kernels. Second it incorporates multiple applications 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. 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.

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 12(a) is composed of eight convolutional layers, five maximum pooling layers and three fully connected layers with total 133 million weight parameters and top-5 error of 10.4% on the ImageNet data set. VGG-16 as shown in Figure 12(b) is composed of 13 convolutional layers, five maximum pooling layers and three fully connected layers with total 138 million weight parameters and top-5 error of 7.4% on the ImageNet data set. VGG-19 as shown in Figure 12(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.

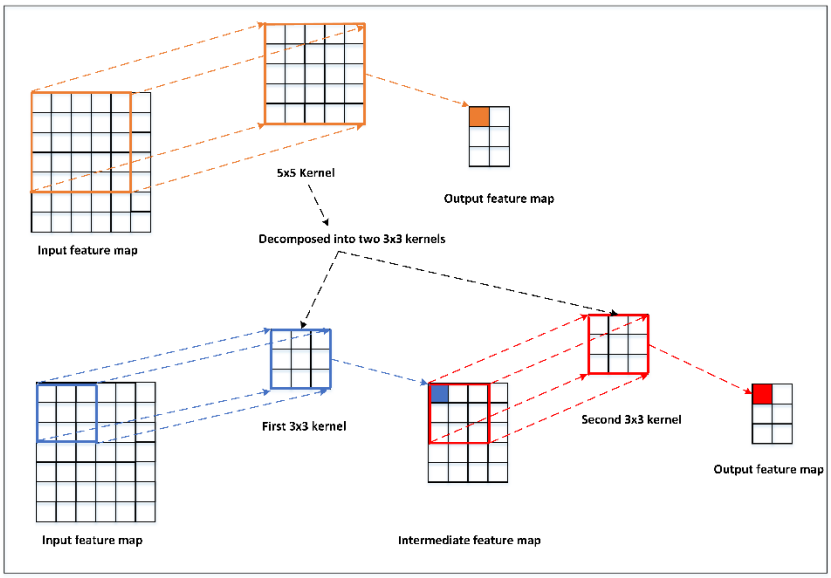

Figure 11 : 5x5 kernel decomposed into two 3x3 kernels

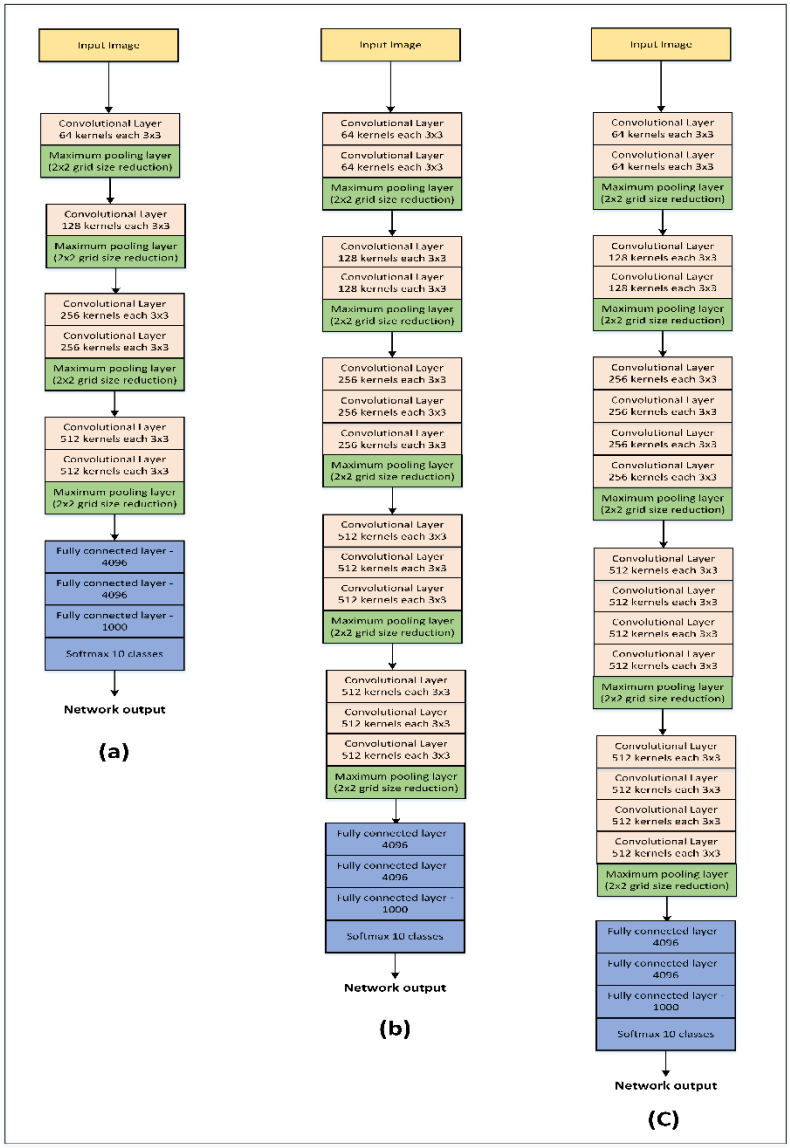

Figure 12 : (a) VGG-11 (b) VGG-16 (c) VGG-19

F. NiN

Network in Network (NiN) [22] 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).

G. GoogLeNet

Also referred to as Inception [23] 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 [24], [25] and [26].

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 13 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.

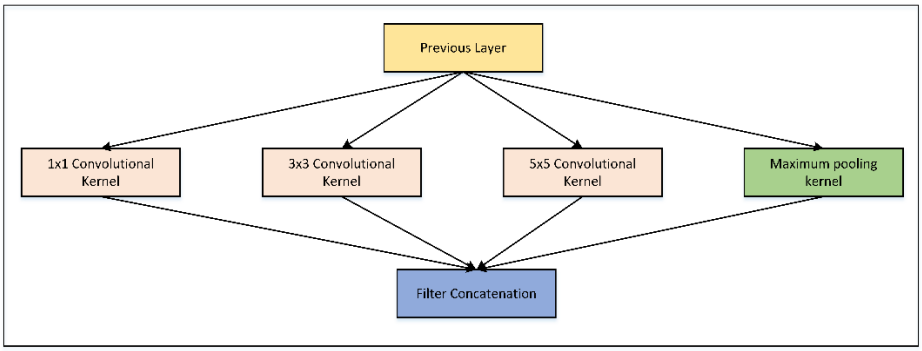

Figure 13 : Naïve Inception module

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 the 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 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, the depth of network is allowed to be increased without a significant computational penalty. Also, these 1x1 convolutional kernels are associated with applying nonlinear function activation enabling them to have a dual propose. Figure 14 shows the inception with dimension reduction.

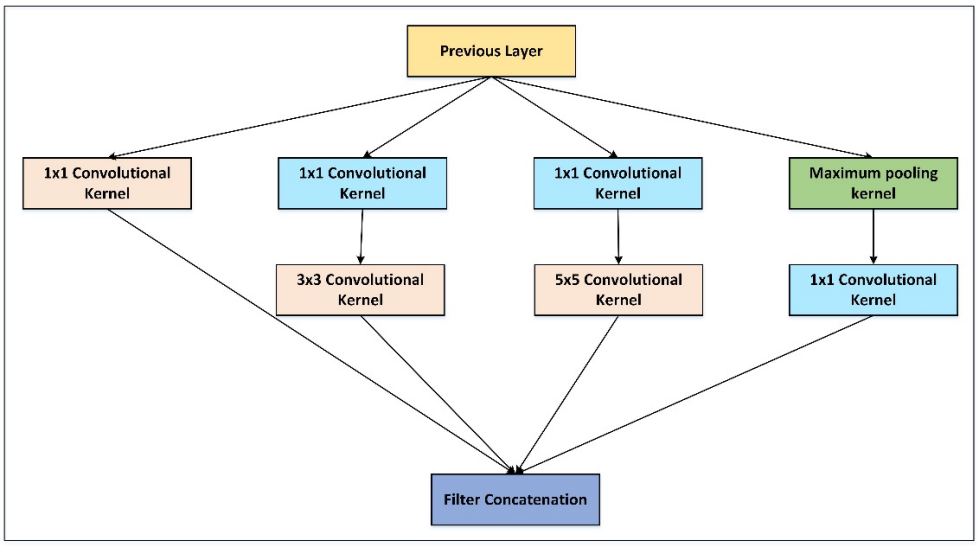

Figure 14 : Inception module with dimension reduction

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.

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 the 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 activations in a mini batch across all locations.

However, one drawback of normalizing each of the inputs of layer is that it may change what a layer can represent, thus batch normalization introduces two learnable parameters; the scale and shift values to ensure that 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) 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 scales 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 dependencies 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 aggregating 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 15 with around 33% computation savings.

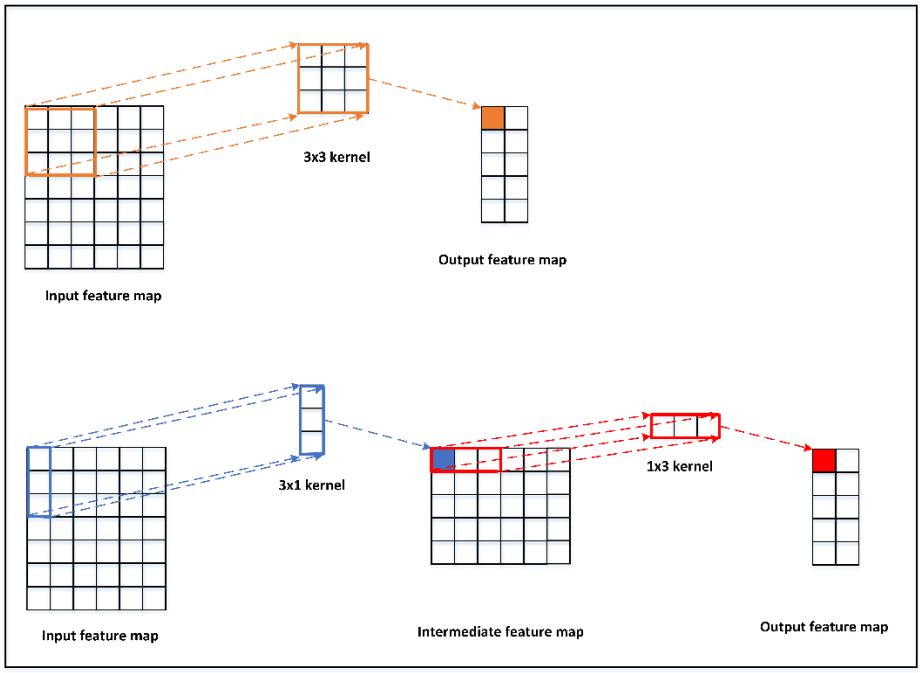

Figure 15 : Decomposing 3x3 kernel into asymmetric kernels

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 16. 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.

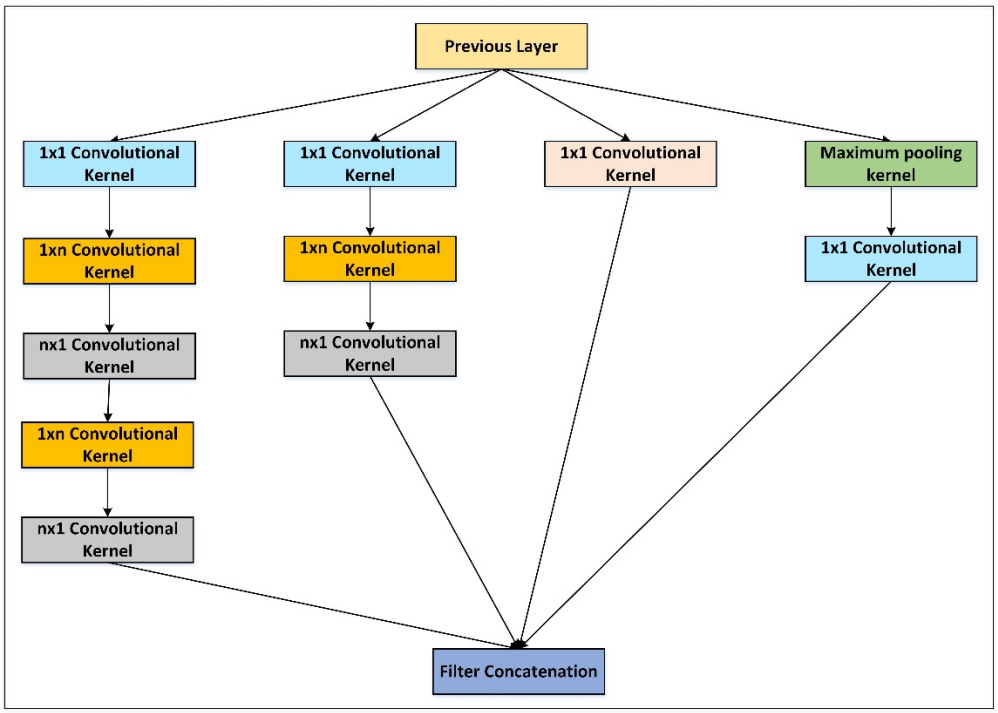

Figure 16 : New Inception module with nx1 and 1xn factorized kernels

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.

4) Fourth version

The fourth version was introduced in 2016 and it introduced a new variant that combines the Inception module with the residual connections introduced in [2] as shown in Figure 17.

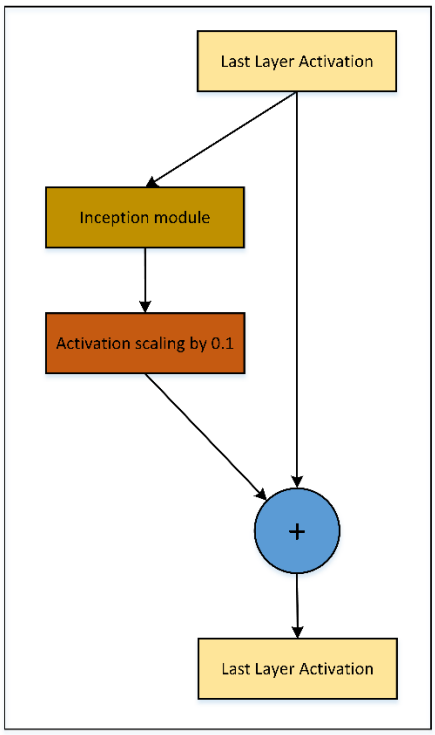

Figure 17 : Inception module accompanied by residual connection

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.

H. ResNet

Also known as Residual Net [2] 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 [27].

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 layers 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 18, 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.

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 19 by replacing the two layers stack with a third 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.

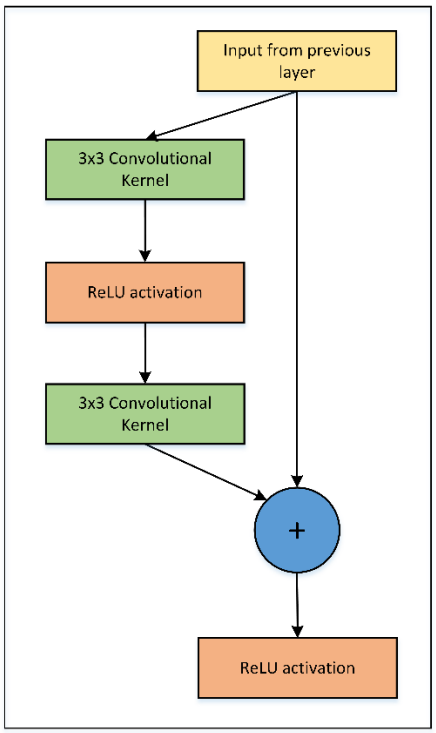

Figure 18 : Shortcut module

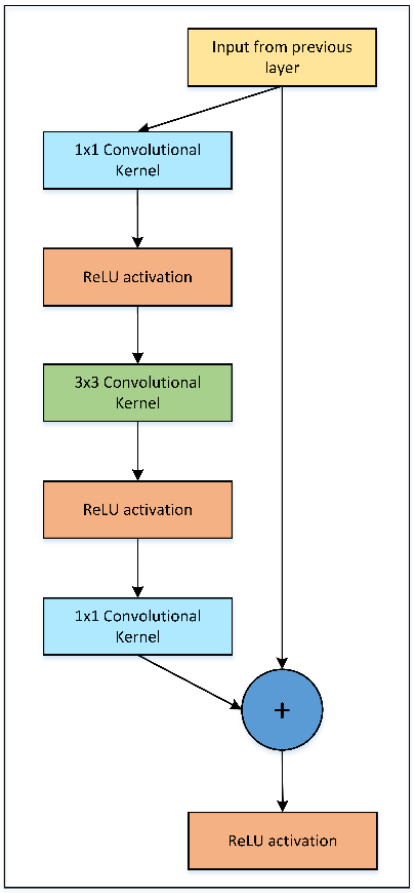

Figure 19 : Bottleneck module

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 parameters 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 parameters and top-5 error of 4.6% on the ImageNet data set.

ResNet-152 is composed of one convolutional layer, 50 bottleneck modules each shall have three convolutional layers and one fully connected layer with total 60 million weight parameters and top-5 error of 4.49% on the ImageNet data set. 56

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 20 (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.

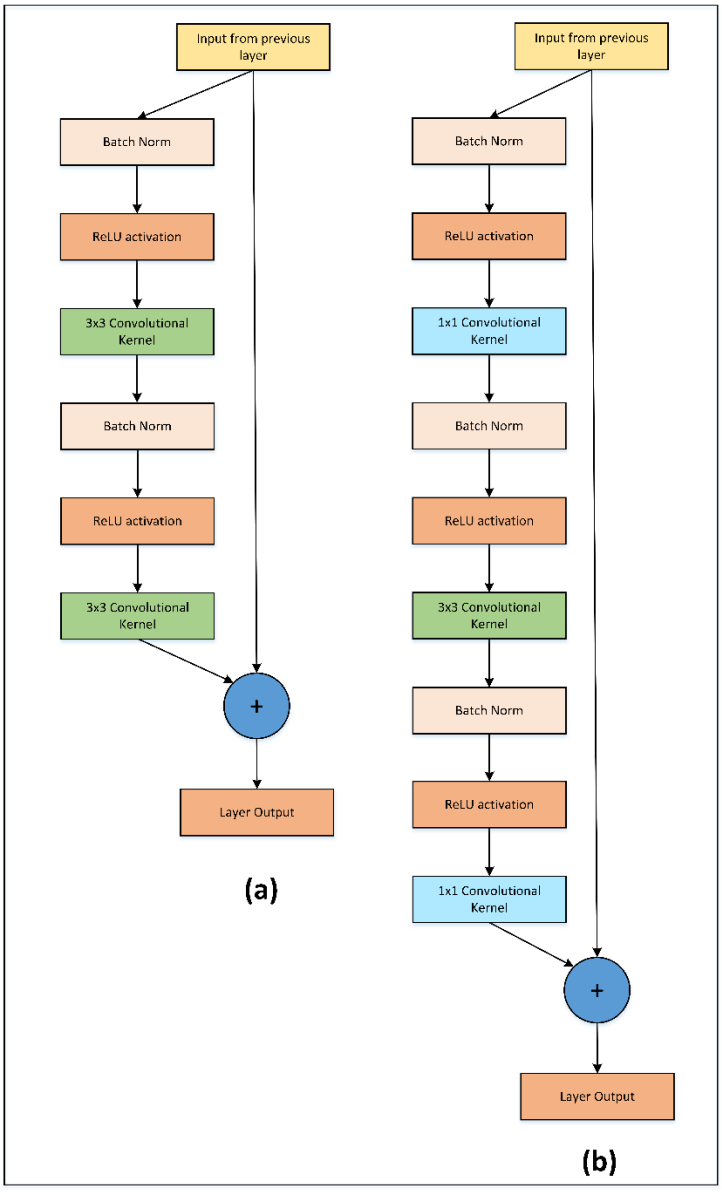

Figure 20 : (a) Modified shortcut module (b) Modified bottleneck module

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

ResNet-152 is composed of one convolutional layer, 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 layer, 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.

I. Observed Trends

Figure 21 summarizes the evolving networks since AlexNet where obvious trends across these networks can be observed

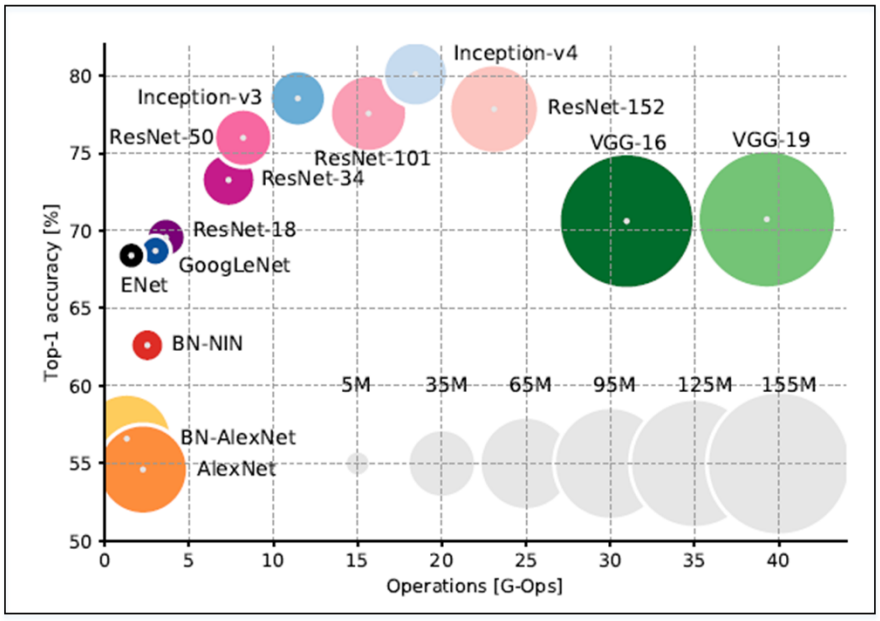

Figure 21 : Different networks compared according to their size, number of operations and Top-1 accuracy from [28]

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 any fully connected layers

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 transformations applied enhancing the network capability to be more discriminative.

4 CNN Optimization Methods

Earlier CNN approaches had considered their figure of merit 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 CNN models and hardware together leading to the evolving of a new figure of merit that is considered with maximizing the accuracy and throughput while minimizing the energy and the cost of hardware infrastructure such that it increases its practical deployment. 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 CNN 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).

A. Reduced precision

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 levels (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 the 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.

1) 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.

i. 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 CNNs can vary in a finer grained manner. For instance, according to [29] 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[30], 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.

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 from BinaryConnect[31] 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 [32] which convert the multiplication and addition operations to XOR operations with 50.42% top-5 accuracy on the ImageNet dataset. Moving to Binary weight nets [33] and XNOR-Nets [33] 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 [34] 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 [35] which allows the weights limit to be 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 [36] 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

ii. 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.

- 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, [37], 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 [38] introduces the Incremental network quantization which divides 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.

- 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, [39] quantizes the weights in a power of two fashion enabling the multiplication operation to be executed as a bit shift operation.

- 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 [40] where it modifies AlexNet to have 256 unique weight values 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.

B. Network pruning

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 [41] 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 finely 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 [42] 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 thought as zero values and can be loosened to include also the near-zero values which encompass more weights and results in more savings without impacting the accuracy. For instance, in [42] the AlexNet number of weights were reduced nine times while maintaining the same accuracy.

1) Area of focus:

Advances in the network pruning focuses on two areas the first is how to efficiently store the sparse weights after pruning which shall need to a compression format to open the benefits of pruning these weights, while the second is 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) Storing Sparse weights

Compressing the sparse weights shall consider how CNN process these weights through the matrix vector multiplication which is one of the fundamental operations within network. There are two compressing formats 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 22 requires the input vector to be read multiple times while each output element is generated once at a time.

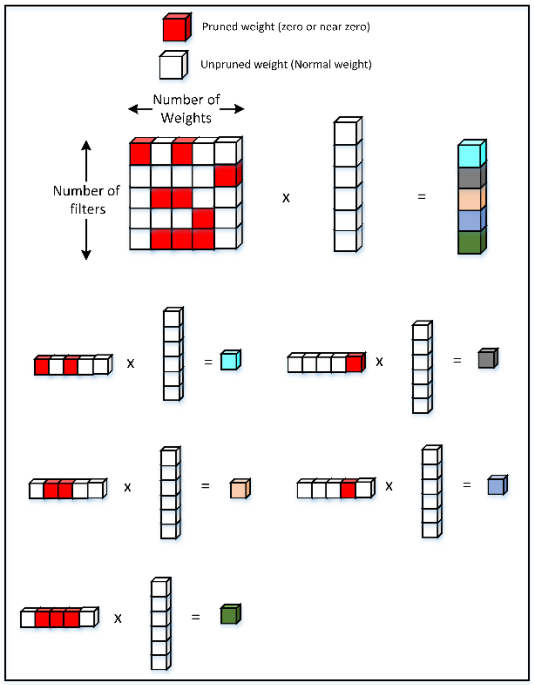

Figure 22 : Compressed sparse row format during matrix multiplication

Meanwhile, compressed sparse column format as shown in Figure 23 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 that 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.

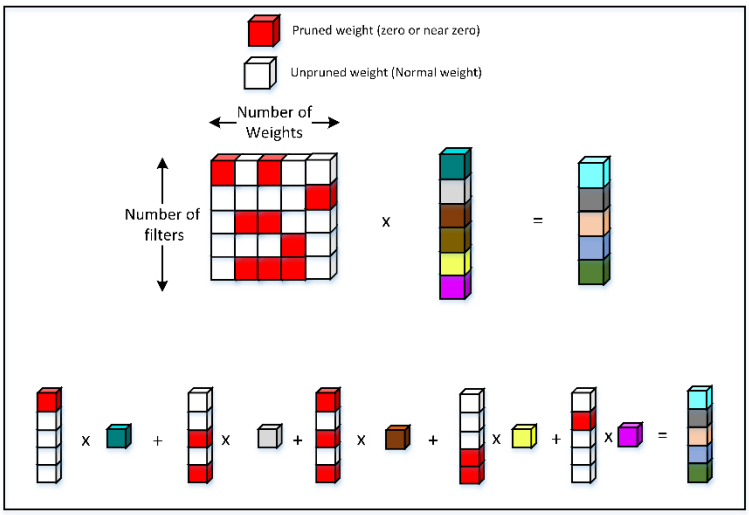

Figure 23 : Compressed sparse column format during matrix multiplication

3) 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 defined criterion which may be a filter entire row or column, a filter channel, 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, Cnvultin[44] which allows dynamically skipping neuron computations if they are below a pre-specified, per-layer threshold.

C. *Activation statistics*

Recently, there are many works 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 24 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 [45] the feature map within the hidden layer of AlexNet can have sparsity between 19% up to 63% depending on the layer.

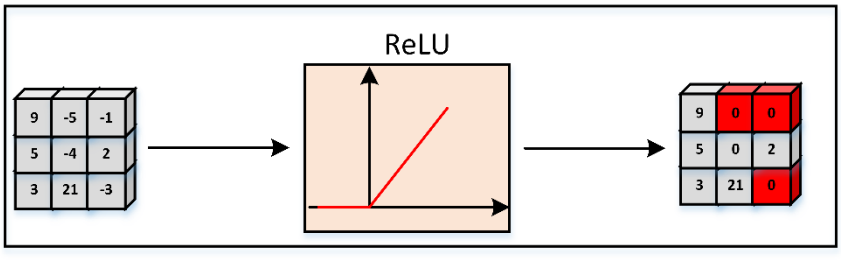

Figure 24: ReLU function

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 that it is predetermined that it is going to fetch a zero value.

Applying this optimization method is used in LRADNN[46] 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[47] 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 exists 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[44] which introduces the Zero Free Neuron Array Format as the compression technique to eliminate the zero activation computations

D. 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 [49], 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.

E. Knowledge distillation

One way to increase the achieved accuracy is to use network ensembles where multiple networks 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 [50] 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 25 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.

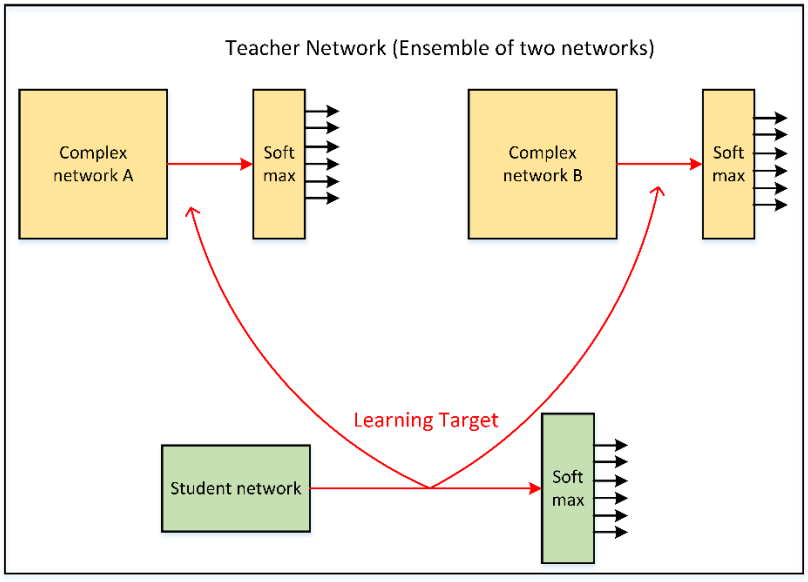

Figure 25 : Knowledge distillation overview

F. 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 [51], Winograd's algorithm [52], Strassen's algorithm [53] and Structural matrix using relaxed Toeplitz form [45].

1) Fast Fourier Transform

Fast Fourier Transform is used to reduce the number of multiplications where the convolution operation is done as a direct multiplication in the frequency domain. As shown in Figure 26 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).

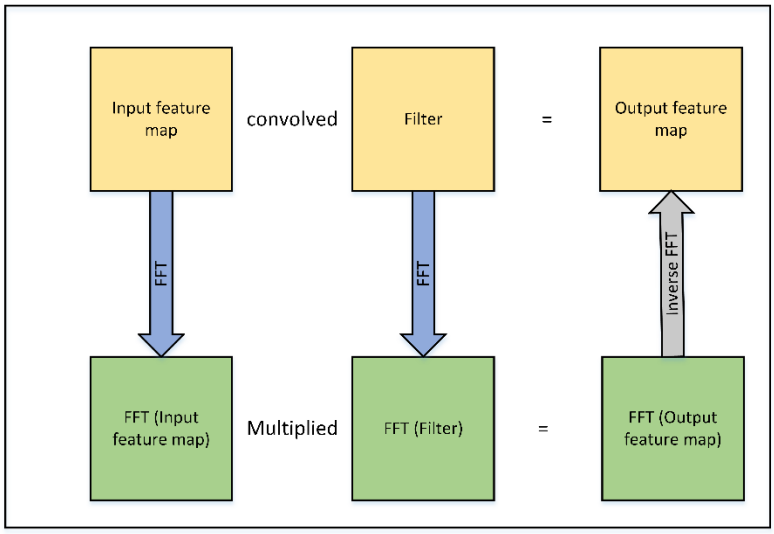

Figure 26 : FFT mathematical transformation

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 helps to reduce the required number of multiplications given that it generates a tile of output elements at each step. It is usually used in smaller filters such as 3x3 where according to [45] it was able to reduce the number of multiplication by 2.25x.

3) Strassen's algorithm

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

4) Structural matrix

Structural matrix using a relaxed Toeplitz form as shown in Figure 27 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.

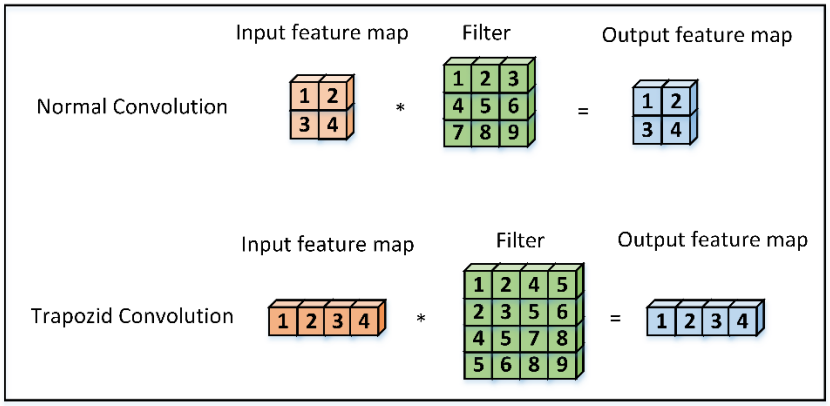

Figure 27 : Structural matrix using a relaxed Toeplitz form

5 CONCLUSIONS

Due to the success of CNN as well as their superior performance when compared to other approaches, they were applied across wide range of applications starting from Image classification to the critical medical diagnosis ones. This paper had introduced briefly CNN keys then moved to survey the progress of the architectures starting from LeNet-5 up to ResNet and finally it had shown different optimization methods that are used to trade-off the accuracy with the hardware complexity in order to achieve more realistic architectures that can be deployed on today's available hardware infrastructure.

BIOGRAPHY

Mohsen Raafat
Mohsen received the B.Sc. in electronics and electrical communications from the Faculty of Engineering, Cairo University, Cairo, Egypt

Prof. Hossam A.H. Fahmy
Prof. Hossam received the B.Sc. and M.Sc. degrees in electronics and electrical communications from the Faculty of Engineering, Cairo University, Cairo, Egypt and the Ph.D. degree in Electrical Engineering from Stanford University, USA

Prof. Mohsen A. A. Rashwan
Prof. Rashwan received the B.Sc. and M.Sc. degrees in electronics and electrical communications from the Faculty of Engineering, Cairo University, Cairo, Egypt, another M.Sc. degree in systems and computer engineering from Carleton University, Ottawa, ON, Canada, and the Ph.D. degree in electronics and electrical communications from Queen's University, Kingston, ON, Canada.

REFERENCES

[1] 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.
[2] 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.

[3] O. Russakovsky et al., “ImageNet large scale visual recognition challenge,” International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, 2015.
[4] 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
[5] R. Girshick, “Fast r-cnn,” IEEE International Conference on Computer Vision (ICCV), Santiago, 2015
[6] 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.
[7] 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.
[8] 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), Boston, 2015.
[9] 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, Columbus, 2014.
[10] "OpenFace,” Cmusatyalab.github.io, 2020. [Online]. Available: https://cmusatyalab.github.io/openface/
[11] L. Wang et al, “Towards good practices for very deep two-stream convNets,” arXiv preprint arXiv:1507.02159,2015
[12] 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, 6, 1155-1166, 2018
[13] 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.
[14] Esteva, et al., “Dermatologist-level classification of skin cancer with deep neural networks,” Nature, vol. 542, no. 7639, pp. 115–118, 2017.
[15] R. Girshick, “Fast r-cnn,” IEEE International Conference on Computer Vision (ICCV), Santiago, 2015.
[16] 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.
[17] Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Advances in neural information processing systems 25, 1097-1105, Nevada, 2012.
[18] C. J. B. Yann, Y. LeCun, and C. Cortes, “The MNIST DATABASE of Handwritten Digits”. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[19] Matthew D Zeiler, Rob Fergus, “Visualizing and Understanding Convolutional Networks,” arXiv preprint arXiv:1311.2901, 2013.
[20] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated recognition, localization and detection using convolutional networks,” arXiv preprint arXiv:1312.6229, 2013.
[21] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
[22] M. Lin, Q. Chen, and S. Yan, “Network in network,” arXiv preprint arXiv:1312.4400, 2013.
[23] C. Szegedy et al., "Going deeper with convolutions,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, 2015.
[24] 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), pp. 448–456, Lille, 2015.
[25] 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.
[26] 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), pp. 4278–4284, San Francisco, 2017.
[27] K. He, X. Zhang, S. Ren and J. Sun, “Identity Mappings in Deep Residual Networks,” arXiv preprint arXiv:1603.05027, 2016.
[28] Canziani, A. Paszke, E. Culurciello,” An analysis of deep neural network models for practical applications,” arXiv preprint arXiv:1605.07678, 2016.
[29] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, “Stripes: Bit-serial deep neural network computing,” 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE, Taipei, 2016.
[30] 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), Long Beach, 2017.
[31] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” In Proceedings of NIPS, Montreal, 2015.
[32] 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.
[33] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks,” European conference on computer vision (ECCV), Springer, Cham, 2016.

[34] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by halfwave Gaussian quantization," Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), Las Vegas, 2017.
[35] F. Li and B. Liu, "Ternary weight networks," arXiv preprint arXiv:1605.04711, 2016.
[36] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," In Proceedings of ICLR, Toulon, 2017.
[37] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "LogNet: Energy-efficient neural networks using logrithmic computations," IEEE International Conference on Acoustics, Speech and Signal Processing *(IICASSP)*, New Orleans, 2017.
[38] 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, Toulon, 2017.
[39] Philipp Gysel, Mohammad Motamedi and Soheil Ghiasi," Hardware-oriented Approximation of Convolutional Neural Networks," arXiv preprint arXiv:1604.03168,2016.
[40] 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, San Juan, 2016.
[41] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," In Proceedings of NIPS, Colorado, 1990.
[42] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," In Proceedings of NIPS, Montreal, 2015.
[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] 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.
[45] 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.
[46] 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.
[47] 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.
[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] 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, San Diego, 2015.
[50] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," In Proceedings of NIPS Deep Learn. Workshop, Montreal, 2014.
[51] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," In Proceedings of ICLR, Banff, 2014.
[52] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," In Proceedings of CVPR, Ohio, 2016.
[53] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," In Proceedings of ICANN, Hamburg, 2014.

تطور الشبكات العصبية التلافيفية : دراسة استقصائية عن بنية الشبكات العصبية و طرق تحسينها

محسن رأفت*[1] ، حسام فهمى*[2] ، محسن رشوان*[3]

*قسم هندسة الاتصالات و الاكترونيات، كلية الهندسة ، جامعة القاهرة، جيزة، مصر**

[1]moh_raafat@hotmail.com

[2]hossam.a.h.fahmy@gmail.com

[3]mrashwan@rdi-eg.ai

ملخص

منذ بداية حقبة الشبكات العصبية التلافيفية تم استخدامها في مجموعة واسعة من مهام الرؤية الحاسوبية مثل تصنيف الصور ، الكشف عن الاغراض و تتبعها و التعرف على الحركة فى المقاطع المصورة. و قد تمكنت هذه الشبكات عند استخدامها من إظهار أداء مذهل و نتائج تفوق نظيراتها من الطرق الاخرى. تستعرض هذه المقالة التقدم المحرز في بيئة هذه الشبكات و الاساليب التى اقترحت لتحسين ادائها و التخفيض من حجمها. كما تختلف هذه المقالة عن العديد من مثيلتها المتوفرة في الأدبيات من تقديم نظرة عامة موجزة عن الميزات الرئيسية للشبكات العصبية التلافيفية متبوعًا بمراجعة التقدم فى بنيتها وأخيراً نظرة عن الطرق التحسين المقترحة فى الادبيات المختلفة التي قد يمكن من نشر هذه الشبكات على البنية التحتية للأجهزة الحالية دون التأثير بشكل كبير على الدقة المحققة.

الكلمات المفتاحية

تعلم الآلة ، التعلم العميق ، الشبكات العصبية التلافيفية