

American International University-Bangladesh

9, 6, 3 Initializer and It's Impact on Convergence

Software Project 2

Supervised by: Dr. Debajyoti Karmaker

Name	ID
Mahmud, Sk. Golam	18-38600-2
Sijan, Asif	18-36804-1
Sarkar, Anik Chandra	18-36514-1
Alam, Md. Shakibul	18-36556-1

A Project submitted for the degree of
Bachelor of Science (BSc) in Software Engineering
at American International University Bangladesh
in December, 2021
Faculty of Science and Technology (FST)

Abstract

Different components make up a neural network made up of deep learning models. The accuracy and performance of a model are determined by a variety of components and niches, including activation functions, optimizers, loss or cost functions, augmentation, dropout regularization, and so on. But an often-overlooked step would be initializing a model's weights and biases. We have presented a framework for weight and bias initialization of a deep learning model in this research. Applying it on different models with the same architecture, with the proposed framework- model's efficiency increased by around 25 to 30 percent compared to initializing random weights and biases. Considering deeper networks, efficiency boost would result in significantly less time to finish training. And the suggested framework will further improve models that rely on efficiency measures, such as models that operate on mobile or embedded systems where resources are limited.

Declaration by author

This project is composed of our original work, and contains no material previously published or written by another person except where due reference has been made in the text. We have clearly stated the contribution of others to our thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in our project. The content of our project is the result of work we have carried out since the commencement of project.

We acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate we have obtained copyright permission from the copyright holder to reproduce material in this project and have sought permission from co-authors for any jointly authored works included in the project.

Approval

The project titled **"9, 6, 3 Initializer and It's Impact on Convergence"** has been submitted to the following respected members of the board of examiners of the department of computer science in partial fulfilment of the requirementsfor the degree of **Bachelor of Science in Software Engineering** on **(14/12/2021)** and has been acceptedas satisfactory.

. . . . . .

Dr. Debajyoti Karmaker
Associate Professor
Department of Computer Science
American International University-Bangladesh

. . . . . .

Abhijit Bhowmik
Associate Professor
Department of Computer Science
American International University-Bangladesh

. . . . . .

Dr. Md. Mahbub Chowdhury Mishu
Assistant Professor & Head (Undergraduate)
Department of Computer Science
American International University-Bangladesh

. . . . . .

Professor Dr. Tafazzal Hossain
Dean
Faculty of Science & Information Technology
American International University-Bangladesh

. . . . . .

Dr. Carmen Z. Lamagna
Vice Chancellor
American International University-Bangladesh

Publications included in this thesis

(No publications included)

Submitted manuscripts included in this thesis

(No manuscripts submitted for publication)

Other publications during candidature

(No other publications)

Research involving human or animal subjects

(No animal or human subjects were involved in this research)

Contributions by authors to the thesis

List of the significant and substantial inputs made by different authors to this research, work and writing represented and reported in the thesis. These could include significant contributions to: the conception and design of the project; non-routine technical work; analysis and interpretation of research data; drafting significant parts of the work or critically revising it so as to contribute to the interpretation.

	Mahmud, Sk. Golam	**Sijan, Asif**	**Sarkar, Anik Chandra**	**Alam, Md. Shakibul**	**Contribution (%)**
	18-38600-2	*18-36804-1*	*18-36514-1*	*18-36556-1*	
Conceptualization	25%	25%	25%	25%	100 %
Data curation	20%	30%	30%	20%	100 %
Formal analysis	30%	30%	20%	20%	100 %
Investigation	25%	25%	25%	25%	100 %
Methodology	55%	15%	15%	15%	100 %
Theoretical derivations	30%	20%	30%	20%	100 %
Preparation of figures	30%	20%	30%	20%	100 %
Writing – original draft	25%	25%	25%	25%	100 %
Writing – review & editing	25%	25%	25%	25%	100 %

Acknowledgments

We would like to express our deepest gratitude to our supervisor, Dr. Debajyoti Karmaker, whose sincerity and encouragement we will never forget. Dr. Karmaker has been an inspiration as we hurdled through the path of this Bachelor's degree. He is the true definition of a leader and the ultimate role model. We are thankful for the extraordinary experiences he arranged for us and for providing opportunities for us to grow professionally. It is an honor to learn from him.

We are grateful for our teachers whose constant love and support keep us motivated and confident. Our accomplishments and success are because they believed in us. Deepest thanks to our external, who keep us grounded, remind us of what is important in life, and are always supportive of my adventures.

Keywords

Weight Initialization, Neural Network, Converge Technique, Computer Vision

Table of Contents

INTRODUCTION ---- **1**

LITERATURE REVIEW ---- **3**

2.1 He Initialization ---- *3*
2.2 Xavier Initialization ---- *3*
2.3 MobileNet ---- *4*
2.4 ResNet-50 ---- *5*
2.5 ResNeXt-50 ---- *6*
2.6 EffNet ---- *7*
2.7 VGGNet ---- *8*
2.8 DenseNet ---- *9*

METHODS ---- **11**

3.1 Number Properties ---- *11*
3.2 Dataset ---- *12*
3.3 Forward Propagation ---- *12*
3.4 Activation Function ---- *13*
3.5 Backpropagation ---- *13*
3.6 Model ---- *14*

RESULTS OR FINDINGS ---- **15**

4.1 Random weight initialization ---- *15*
4.2 9, 6, 3 Custom Weight Initialization ---- *16*

DISCUSSION ---- **17**

CONCLUSION ---- **18**

REFERENCES ---- **19**

List of Figures

FIGURE D. 1 : AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH (AIUB) ---------- 36
FIGURE D. 2: EXAMPLE OF PLACING IMAGES SIDE BY SIDE---------- 38

List of Tables

TABLE E. 1: AIUB CURRENTLY OPERATES UNDER FOUR DISTINCT FACULTIES X---------- 41
TABLE E. 2: ROW SPAN EXAMPLE ---------- 41
TABLE E. 3 : TEST TABLE---------- 41

List of Abbreviations and Symbols

Mention all the abbreviations and the different symbols that is used in this document.

Abbreviations	
CS	Computer Science
CSE	Computer Science and Engineering
NN	Neural Network
HCI	Human Computer Interaction
NLP	Natural Language Processing
etc.	*etc.*

Symbols	
$\hat{\rho}$	Density operator
®	Convolution
etc.	*etc.*

Chapter 1

Introduction

Neural network's concept is found as the biological human brain's neurons are connected with each other. Perceptron is a concept of a single neuron. Sensory data is interpreted through neurons. Neural networks are also known as Artificial Neural Networks (ANN). The data may be numerical, images or vector signs, music etc. Neural networks are used for this type of data's classification, properly labeling the data properties and identifying any real world object. For grouping neural networks, cluster similar types of data based on their attributes, properties. Neural network algorithm is a combination of reinforcement learning, classification and regression. It solves real world problems- prioritizing the customers of a shop, managing big data and data scientists excavating the important data for their research. There are two types of learning- a) supervised learning b) unsupervised learning. If the model learns from the labels this is called supervised learning and if it learns from data's characteristics not from labels it is called unsupervised learning.

It is a combination of input layers, hidden layers, and output layers. Layers are made of nodes. Nodes have some properties like weight, learning rate and coefficients. The input layer sends the input signals to the next layer. These inputs have a summation of weights and are passed with an activation function. A neuron is activated if it passes a signal through it. Hidden layer performs the computation task and at least one hidden layer exists in a neural network.[1] A neural network has a multi-layer perceptron and those are hidden layers. Perceptron neutrons are the primary unit for the perceptron layers. Calculations are composed of weights of the neurons multiplied by the inputs and threshold value. These weights are adjusted after the loss calculation; then inputs get a perfect weight. In output layers, the activation function activates the perfect match result.[1]

The output layer is subsequently dependent on the input layer. Neuron acts like a switch when it turns on the time of feed forward. Network depth depends on the number of hidden layers. One network can be deep learning if it has more than 03 layers including input, output. In training, a unique set of features are used to track in feature maps. Features are recombined from previous layers and lastly it has turned into a complex feature. A dataset can contain millions of data and each data has many parameters. In the world a large volume dataset is present and they are unlabeled. To organize and cluster the data neural network is the solution. The process is called feature extraction.

The weights of the neurons are not properly tuned. With the help of backpropagation weights are fine tuned. Backpropagation is a process where in model training, previous epochs generate error rates and weights are fine-tuned from the learning of errors. In back propagation it calculates the gradient of the loss function.and it computes one layer at a

time. For loss calculation in a neural network has introduced many functions like- mean squared error,
Cross entropy loss. The stochastic gradient descent algorithm is used for weight optimization, to minimize the loss it incrementally changes the network weights. It needs a starting point to gear up the process. Different approaches for weight initialization- Sigmoid, Tanh. Both are called xavier initialization. In this process uniform between the range of negative or number of neurons square root to positive. Weights can be wider or narrow, it depends on inputs. With fewer inputs, the bound of weights becomes wider and for more inputs the bound is narrow. [2] Issues have been found in weight initialization in *xavier;* ReLU is a solution as another activation function. This is a modified approach for weight initialization and is mostly used in hidden layers of multilayer perceptrons. ReLU is the currently standard approach for weight initialization and is called *he* initialization. The creator of this approach is Kaiming He. "*He"* initialization is a measured probability between 0.0 and the square root of *2/n;* 'n' for input node numbers. With this initialization, weights initialization is more accurate.

Convolution neural network is a specialized type of neural networks. This model is specially designed for 2D image data. This is named after it's layer name and as convolution layers do the main trick here. The base structure is like neural networks and the calculation process also. It calculates with input and weight multiplication. There is a filter for extracting the features and dot product between inputs and feature is the process for feature calculation. If a filter has a purpose for a specific feature, it searches the input across this feature.

After the convolutional layer there is a pooling layer. The goal of the layer is to decrease the feature map size to improve in computational timing, less computation. From the feature map the largest element is selected for max pooling. Average pooling is used for elements of average. Sum of the element is calculated in sum pooling. After the pooling layer it connects to fully connected layers. This layer is used for connecting the neurons of different layers. All these layers are present before the output layer. In fully connected layers, the flatten images from previous layers are fed to this layer. Then mathematical operations begin for classification.

There is an issue of overfitting of the training model; it can happen due to any particular model performing well on a dataset. But it has a negative impact for the performance of the model. Dropout is a solution for the problem of overfitting. Numbers of neurons can be dropped during the training process and it reduces the size of the model. Neurons are dropped randomly. There are various good CNN architectures at present. VGG16, AlexNet, MobileNet are common and popular CNN architectures.

Chapter 2

Literature review

The initialization of parameters is the first step to consider while establishing a neural network; if done correctly, optimization will be performed in the shortest time feasible; otherwise, converging to a minima through gradient descent would be impossible. Most common practice would be randomly initialized weights and biases initialized with 0. Incase of random weight initialization, in terms of high valued random weight, dot products become significantly higher. After applying activation functions like sigmoid, the resulting value lurks around 1 where gradient slope varies slowly and the network takes a long time to complete learning. Taking low values into account, activation functions map it to 0 resulting in the same outcome.

To solve this problem for different activation functions, other initialization techniques are required and that is the reason behind originating various initialization strategies like Xavier initialization, He initialization, and so on.

2.1 He Initialization

Despite being the usual method for initializing weights in Sigmoid and Tanh activations, Xavier initialization has been discovered to have issues with rectified linear (ReLU) activation functions. He initialization is the current standard method for initializing the weights of neural network layers and nodes that employ the rectified linear (ReLU) activation function. Unlike Sigmoid and Tanh, Relu is not concerned by vanishing and exploding gradients. Although it aims to achieve the same, He initialization differs from Xavier initialization. This distinction is due to the nonlinearities of the ReLU activation function, which render it non-differentiable at x=0. While Xavier initialization strategy performs random initialization with a variance of 1/N where N is the number of input neurons to a certain layer, He initializer does the same operation with 2/N variance. While comparing two methods on extremely deep models of up to 30 layers, He initializer was able to make the extreme deep model converge while Xavier method fully halts the learning process.[1]

2.2 Xavier Initialization

"Glorot" or "Xavier" initialization is the present widely used methodology for initializing the weights of neural network layers and nodes that employ the Sigmoid or TanH activation function. The Xavier initialization strategy is computed as a random number with a uniform probability distribution (U) between -(1/sqrt(n)) and 1/sqrt(n), where n is the number of node inputs. And in the normalized Xavier initialization, a random number

with a uniform probability distribution (U) between -range -(sqrt(6)/sqrt(n + m)) and sqrt(6)/sqrt(n + m) is taken. These approaches are useful for initialization because they reduce the likelihood of exploding or vanishing gradients. They selected the weights to be neither too large nor too small than one. As a result, the gradients do not disappear or burst too soon. They aid in avoiding delayed convergence while also preventing us from fluctuating off the minima.[2]

2.3 MobileNet

2.3.1 MobileNetV1

MobileNets[3] are low-latency, low-power models that were created for usage with mobile applications and to meet other hard-ware requirements, such as embedded vision applications. To dramatically minimize the computational cost, it employs Depthwise Separable convolutions introduced in [9]. Two phases are involved in depth-wise separable convolutions. Firstly, there is a Depthwise convolution layer that applies a single filter to each input channel, followed by a 1x1 Pointwise convolution layer which combines the output. If compared against traditional convolutional operations, this reduces the number of parameters by DK2, Where DK is the kernel size. If a 3x3 kernel is taken, then the number of parameters are reduced 9 times compared to a traditional convolution layer.

Mobilenets also introduced two hyper parameters to make existing Mobilenets faster for certain use-cases where memory and power consumption is more restricted. One of them is α, width multiplier- which shrinks the kernel size. Another one is ϱ, resolution multiplier- which reduces the representations. This results in slightly less accurate but faster networks.

2.3.2 MobileNetV2

MobileNet V2[4] was an upgrade over its predecessor, MobileNet V1. In MobileNet version 1, the main block was a Depthwise separable block with a Depthwise convolution followed by a Pointwise convolution. MobileNet v2 now includes bottleneck residuals blocks. An expansion layer (1x1) is used, followed by a depth convolution layer (3x3) and a projection layer (1x1) (1x1) In the Expansion layer, there are more output than input channels. There is also a hyperparameter called the expansion factor, which indicates how much the data will be enlarged in this case. The Expansion factor is set at 6 by default. Using the setting 6, the convolution layer will treat a 24-channel tensor as if it was a 144-channel tensor. When it's time to project, it'll convert the channels back to 24 again. A high-dimensional tensor is used for filtering in between the low-dimensional tensors. The residual link, which is comparable to ResNet and aids with gradient flow, was also incorporated in Mobilenet v2's construction block (through the network). When the number of input and output channels are varied in a block, this option is only used. Normalization is done in batches for each layer, and the activation function is Relu 6[7].

MobileNet V2 consists of 17 residual blocks in a row, followed by a regular 1x1 convolution, a global average pooling layer, and a classification layer, respectively.

2.3.3 MobileNetV3

Next generation of MobileNets are based on combination of complementary search techniques and novel architecture designs. MobileNetv3[5] added squeeze and excitation layers on top of the building blocks taken from MobileNet v2 with including further treatment such as NAS and NetAdapt. It used h-swish nonlinearity, dropping relu-6 from MobileNet v2. When constructing the output feature maps, the squeeze and excitation modules help by assigning unequal weights to distinct channels from the input rather than the equal weights that a typical CNN assigns. Squeeze and excitation are usually introduced to the resnet/inception blocks individually. Similar to MnasNet[8] which was built on top of MobileNet v2, MobileNet v3 used platform-aware NAS for block-wise search followed by NetAdapt[10] algorithm to search per layer for number of filters. Newly introduced h-swish represents a modified version of swish nonlinearity which is faster to compute and more quantitatively oriented. MobileNet v2's inverted bottleneck were redesigned to reduce latency and preserver high dimensional features. Instead of previous 7x7 spatial resolution, they were now computed as 1x1 spatial. This resulted into an decreased latency of 7 ms, 11% of the running time and number of operations by 30 million MAdds without any accuracy loss. The initial set of filters is another pricey layer. To generate initial filter banks for edge detection, most mobile models now use 32 filters in a full 3x3 convolution. These filters are frequently mirrored versions of one another. Redundancy was reduced by reducing the number of filters and experimenting with different nonlinearities. Hard-swish nonlinearity was selected for this layer because it outperformed other nonlinearities examined in terms of performance and reliability Although the number of filters was reduced, the accuracy remained the same as with 32 filters using either ReLU or Swish. 2 milliseconds and 10 million MAdds are saved as a result. On the experiment section, MobileNet v3 behind SSDLite[6] performed 27% faster than MobileNet v2 and 30% faster than MnasNet on COCO test set. In terms of Semantic Segmentation, MobileNet v3 were slightly faster as a backbone than v2s (around 10%). Use of Neural Architecture by Google yields the best possible results. Such network search techniques can be a fantastic motivator to use and develop deep learning and computer vision algorithms in the future.

2.4 ResNet-50

Residual Network is the full form of ResNet[11] architecture and this model introduces us with the residual learning which is the main concept of this model. Residual learning simply learns as subtraction of feature learned from input of that layer. This residual learning resolves the problem of degrading accuracy on a deep neural network. This residual network makes it possible to train ultra deep neural network without sacrifice the accuracy. ResNet have different types depending on the layers, here we explain the ResNet50 which have 48 convolutional layers. Beginning it have a convolution layer with kernel size 7*7 and 54 different kernels all with a stride size 2. After first layer there will

be a max-pooling layer this max-pooling layer contain (3*3) kernel with stride 2 pixels max-pooling layer applied to reduce the input resolution. In the next convolution layer, there is 1*1, 64 kernel following this 3*3, 64 kernel and lastly 1*1, 256 and these three layers are repeated three times and it gives us 9 layers here. Next layer 1*1, 128 kernel following this 3*3, 128 kernel and lastly 1*1, 512 and these three layers are repeated four times and it gives us 12 layers here. After that there have kernel of 1*1, 256 kernel following this 3*3, 256 kernel and lastly 1*1, 1024 and these three layers are repeated six times and it gives us 18 layers here. Last convolution layer has kernel of 181, 512, 3*3, 512 and 1*1, 2048 and also these three layers repeated three time and these convolution layer gives us 9 layers. At the very last stage of the implementation of the architecture an average pooling layer is applied for the replacement of fully connected layer. This average pooling layer have no parameter to optimize this layer and this layer also help to reduce the complexity of the model. At the last layer this model has a fully connected layer contain 1000 nodes and after that a softmax function applied. We can't count the activation function and max/average pooling here without those layers this model has 50 layers. This model has 25.5 million parameters.

2.5 ResNeXt-50

In 2017 a model has proposed name ResNeXt[12] and it is a homogeneous neural network. This model proposes to reduce the hyperparameters. Achieving the goal, they used "cardinality", an additional dimension on top of the width and depth ResNet. This ResNeXt-50 model have some similarity with the ResNet-50 architecturally they look very similar but the main differences in residual block. This model building block that aggregates a set of transformation with the same topology. The great side of this model is it can able to increase classification accuracy by increase of cardinality where they have to maintain complexity under restricted condition. Residual block of ResNeXt is followed by two simply rules, i) If the blocks produce same-dimensional spatial maps, they share the same set of hyperparameters. ii) Each time the spatial maps are down sampled by a factor of 2, the width of the blocks is multiplied by a factor of 2. A basic ResNeXt model follow the split-transformation-marge strategy where cardinality is 32 and dimension is 4 also for the transformation used aggregate transformation,

here, C = cardinality, T= arbitrary function.
Inside of a block when input is come firstly it split in to 32 group every group have same dimension then transform the groups and lastly merge or concatenate those group and the input is also added with the output. Rest of the architectural process are same as ResNet-50 that we are learned previously. ResneXt-50 model perform better then ResNet-50. ResneXt-50 have 25 million parameters also it has less parameter as compared to ResNet-50[11].

2.6 EffNet

EffNet [13] is optimized and created to solve the issue of MobileNet [14] and ShuffleNet [15]. The Targets of recent CNN architectures are running efficient algorithms on cost effective hardware. Pruning trained networks, fix-point conversation of 32bit networks are examples of efficient solutions. In vanilla convolution layers four dimensions are present - row, columns, channels in, channels out and those help to scale the computational cost. MobileNet [14], ShuffleNet [15] are separate computations but are efficient in large volume data sets. In slimmer datasets both performances are bottlenecks.

They proposed alternative solutions that re-train the proportional decrease to step up the accuracy level. They improve it by optimizing the data flow and neglecting practices. Their architecture allows them to deploy large scale networks in low capacity hardware devices.

Also other related algorithms target optimization techniques and proposed Bayesian optimization framework like- maximize the probability for increasing accuracy. Others optimize computational complexity by computational complexity. Applied reinforcement learning on LSTM [16] for optimization and improving accuracy. But it requires more steps and complicates the process. Another proposed pruning algorithm for better accuracy with minimal cost, but it has some issues like- requires additional phase, if the network has changed and requires fine tuning. EffNet [13] mode could be converted to fix-point, pruned and optimized for the best set of hyper parameters.

After reducing computation cost, the algorithm allows the redistribution of computation power to more critical places. But this type of modification addresses some issues like bottleneck structure, strides and pulling, separable convolutions, residual connections.
The bottleneck structure reduces the 08 input channel to the output channel. They noticed dropping off accuracy after moderate reduction proposed to use bottleneck factor of two. It was fruitful because the first depthwise convolution layer doubles the amount of channels. They found some issues when MobileNet and ShuffleNet use stride two. They suggest using separable pulling where first apply a 2x1 pulling kernel after the first spatial convolutional layer.

They designed an efficient convolution of blocks to solve the issue of data compression. For saving competitions in the second layer, they have split the 3 X 3 depthwise convolution into two spatial layers. After the first depthwise convolution, the 1 X 2 Max polling proposed architecture performs better than MobileNet, ShuffleNet [15] by getting accuracy of 80.2% in CIFAR10 dataset. Same performance available in Street View House Numbers, German Traffic Sign Recognition benchmark. The focus of this architecture was on embedded devices and mobile hardware. [13]

2.7 VGGNet

VGG16[17] architecture was proposed in 2014 by Karen Simonyan and Adrew Zisserman. This model was presented in competition of ILSVRC(ImageNet Large Scale Visual Recognition Challenge). The goal and achievement of this architecture is getting more accuracy on model training and testing with a deep dense convolutional network. It is possible when they used a small filter 3x3 in every convolutional layer.

ILSVRC targets two things- 1. Object localization 2. Image Classification. This VGGNet performs well in both criteria. Except the datasets of ILSVRC, in other image recognition datasets their proposed VGGNet achieve excellent performance even without fine tuning of parameters.

The configuration of this architecture is based on the depth of ConvNet and they used the same principles in all ConvNet layers.

In training they use fixed-size 224x224 RGB images. When the images pass through a stack of convolutional layers a smallest 3x3 filter is used for extracting features. The value of stride is fixed to 1. Padding is 1 for 3x3 conv. layers. Spatial padding is used in max pooling layers. In max pooling layers, 2x2 filter is used with stride 2.

After the conv. layers and max pooling layer there are 03 fully connected layers. First of two layers have 4096 channels and the last one has 1000, because there are 1000 categories in ImageNet. In all hidden layers, ReLu is used and in the final layer softmax function is used for categorizing images.

In this VGG architecture different depth layers were proposed- 11 weight layers to 19 weight layers. Layers of width increase from 64 to 255 at max level.

Key difference between the previous top proposed model in ILSVRC and VGG16 is the small size filter and stride. Previously 11x11 or 7x7 filters were used in architecture. But in VGGNet Simonyan and Adrew Zisserman have used 3x3 filter and stride 1. As a result, the decision function becomes more discriminative. Secondly, the amount of parameters has been decreased, when the 3x3 filter is using 81% parameter has been decreased.

Small size filter has been used before, but the amount of layers was not deep and for evaluation a large scale image dataset was not present like ImageNet.

VGG16 has 16 layers, a combination of convolution layers, max pooling layers and fully connected layers. There are 05 blocks and each block has a max pooling layer, rest of the layers are convolutional layers. Last 03 layers are fully connected layers.

VGG19 performs slightly better than VGG16. But it consumes more memory because of extra layers. The last 03 blocks present more convolutional layers in this architecture. [17]

In recent years the use of high performing computers has increased and there are large scale image repositories like IMAGENET, DENG. Combination of both helps in research and development in CNN. In ILSVRC(ImageNet Large Scale Visual Recognition Challenge) everyone tries to encode the shallow feature from a high dimensional image.

In VGGNet after adding extra layers, 3x3 filters in every layer creates large network depth. This deep density raises issues like – model performance drop, failing to get good network accuracy. This type of degradation proves that all models are not equivalently optimized. One solution is proposed in this paper of this degradation problem – Residual Learning.

Residual creates a shortcut connection with the CNDS network. CNDS is a 08 layer structure. Their experiment shows that combination of Residual-CNDS enhances the accuracy of the CNDS network.

Late research focuses on increasing accuracy on computer vision datasets. After trial with multiple architectures, accuracy level depends on some parameters and these parameters have some advantages.

For small size models (equal to 10MB or less) FPGA deployment becomes feasible. But if the application on client side OTA(over the air) may be required and it requires large data.

They compressed the main VGGNet architecture by reducing parameters and they proposed an architecture ResSquVGG16[19] with equivalent accuracy of VGG16. This ResSquVGG16 is smaller and faster than VGG16. It resolves the issue of degradation and makes improvement in time and size. Applying the Iandola compression method in this architecture improves without any modification.

In depth network residual learning helps to improve accuracy by decaying the degradation. They skipped some layers in shortcut connection using a formula $F(x) = H(x) – x$ to stop degradation. This shortcut connection has advantages – 1. Parameter free, combine with gating function 2. It is the option of optimization through stochastic gradient descent. By using open source deep learning datasets, this shortcut connection has been implemented.

Iandola proposed an architecture for high accuracy with a small number of parameters called SqueezeNet. He follows 03 criteria, 1. 1x1 filters will take the place of former 3x3 filters. 2. Number of 3x3 input channels will be reduced. 3. Down sampling for large activation maps. All criteria help to get good accuracy by reducing parameter count.

VGG16 has 13 convolutional layers and 03 fully connected layers. But ResSquVGG16 contains 12 fire modules and 04 convolutional layers. A Scale layer has been added in every fire module and first convolutional layer. Size of the kernel is 3x3 and stride is 02. [18]

2.8 DenseNet

In recent years CNN has become increasingly deep and as a result much information vanishes and washes out at the endpoint. ResNets [20], Highway network and FractalNets [21] solve this deep layer problem by creating short paths between the layers. In this paper their proposal was making connections with each layer directly by matching feature map

sizes. There is an input which forwards the feature-maps from the previous layer to the next layer. In this architecture, features are combined by concatenating instead of summing up. In L layer network the number of connections are . This dense network requires fewer parameters than other CNN architecture. In traditional feed-forward architecture states are moved from one layer to another layer. ResNet [20] prevents information exploitation by additive identity transformation. It shows that many layers drop during training time. They proposed DenseNet [22] between information that is added and information that is preserved. DenseNet [22] has 12 filters in every layer. Addition of small feature maps of this architecture does not affect the network. Its feature maps help as a final decision maker. DenseNet has better parameter efficiency, improved gradients and flow of information. Its regularizing effect reduces over-fitting on smaller size datasets. ResNet uses bypassing paths for optimization without any difficulties like HighwayNets [23]. Stochastic depth drops layers randomly for improvement of network training and this is proposed for use in ResNet. This performs well with deeper networks and exploits the network to reuse feature maps.

DenseNet supports batch normalization which explains that the L numbers layers network has non-linear transformation. The lth layer receives the feature maps from preceding layers. The inputs are feature maps(x) and Hl for layers. Features maps are collective multiplication with layers and it creates a very dense network. ReLu is the activation function and 3 X 3 is the convolution layer. For simplicity, dense networks are divided into dense blocks. One of those, transition layers normalize layers.

It produces a bottleneck layer which reduces the number of input feature maps to help improve the computational efficiency. This bottleneck layer consists of batch normalization, relu and convolution layers (1x1 and 3x3). In compression feature maps are reduced at transition layers. They made experiments on CIFAR-10, ImageNet, SVHN. ResNet, FractalNet has more depth than DenseNet. But the error percentage of DenseNet(5.92-7.00) is less than those of ResNet(11.33-11.66) and FractalNet(7.33-10.18).

They introduced the direct connection with layers of the same feature map size in their DenseNet. It is optimized with scaler features. It requires less parameters. They plan to use DenseNet as a feature extractor in the future. [22]

Chapter 3

Methods

To build a neural network, concept of number pattern can be used for converging epoch. Here we have used digital root of some number to make weight initialization more efficient for backpropagation.

3.1 Number Properties

There is a plethora of number properties that has already proven for various number pattern. For instance, if we keep doubling number starting from 1 and calculate digital root of each number, we can generate a pattern consisting of 1,5,7,8,4,2 repeatedly. Here is an example.

Number (Starting from 1)	Binary root
1	1
2	2
4	4
8	8
16	(6+1) = 7
32	(3+2) = 5
64	(6+4) = 10 = (1+0) = 1
128	(1+2+8) = 11 = (1+1) = 2
256	(2+5+6) = 13 = (1+3) = 4
512	(5+1+2) = 8
1024	(1+0+2+4) = 7
2048	(2+0+4+8) = 14 = (1+4) = 5
4096	(4+0+9+6) = 19 = (1+9) = 10 = (1+0) = 1
8192	(8+1+9+2) = 20 = (2+0) = 2
16384	(1+6+3+8+4) = 22 = (2+2) = 4
32768	(3+2+7+6+8) = 26 = (2+6) = 8
65536	(6+5+5+3+6) = 25 = (2+5) = 7
131072	(1+3+1+0+7+2) = 14 = (1+4) = 5

Subsequently, the same pattern can be generated from halving numbers starting from 1.

On the other hand, if we start from 3 or 6 and follow the same procedure, we will get either 3 or 6 repeatedly.

Number (Starting from 3 or 6)	Binary Root
3	3
6	6
12	(1+2) = 3
24	(2+4) = 6
48	(4+8) = 12 = (1+2) = 3
96	(9+6) = 15 = (1+5) = 6
192	(1+9+2) = 12 = (1+2) = 3
384	(3+8+4) = 15 = (1+5) = 6
768	(7+6+8) = 21 = (2+1) = 3
1536	(1+5+3+6) = 15 = (1+5) = 6

Starting from 9 will always results 9.

Number (Starting from 9)	Binary Root
9	9
18	(1+8) = 9
36	(3+6) = 9
72	(7+2) = 9
144	(1+4+4) = 9
288	(2+8+8) = 18 = (1+8) = 9
576	(5+7+6) = 18 = (1+8) = 9
1152	(1+1+5+2) = 9
2304	(2+3+0+4) = 9
4608	(4+6+0+8) = 18 = (1+8) = 9
9216	(9+2+1+6) = 18 = (1+8) = 9
18432	(1+8+4+3+2) = 18 = (1+8) = 9

Therefore, the digit 3, 6 and 9 has a special property that is repetition capability.

3.2 Dataset

To implement our concept, we have used Animal Image Dataset from Kaggle where consists 3 different types of classes which are cats, dogs and pandas each have 1000 images. We have considered total 2850 images for training set and 150 images for test set. For our working flexibility we have resized the image into 60*60 and stored it into an array to reduce computational complexity [25].

3.3 Forward Propagation

We have initialized weight for each convolutional layer where the weight of each layer has random distribution ranging from 0 to 1 and divided the digital roots from the numbers that has repetition capability. For each perceptron, the inputs X_1, X_2, …., X_n flows through their corresponding edge and the weight that we have initialized got multiplied with the inputs. It can be expressed as:

$$Y = \sum_{i=1}^{n} W_i X_i + b$$

Where, 'Y' is the output and 'b' is bias. We have used some predetermined bias value for each convolutional layer to ensure the weighted sum non-zero. For multiple perceptron, the weight and input are considered as matrix and they perform dot product to reduce computational complexity.

$$h_1 = W_{11}*X_1 + W_{12}*X2 + W_{13}*X_3 + b_1$$
$$h_2 = W_{21}*X_1 + W_{22}*X2 + W_{23}*X_3 + b_2$$
$$h_3 = W_{31}*X_1 + W_{32}*X2 + W_{33}*X_3 + b_3$$
$$O = W_{O1}*h_1 + W_{O2} * h_2 + W_{O3}*h_3 + b_O$$

Here, h stands for hidden layer neuron and O stands for output layer.

3.4 Activation Function

The term 'Activation Function' implies a function that makes any neural network stronger by helping it learning various complex patterns from the input data [24]. In a multi-layer neural network, the activation function takes input from the previous neuron and convert it in some form (depending on which activation function) to make a suitable output for passing it into another neuron. There are several activation functions including Softmax, Tanh, ReLU, Sigmoid etc.

Activation functions helps to take the value of the output to a specific limit as per our requirement. Activation functions receives input as $W*X + b$ where W is the weight, X is the inputs and b is the bias. In this project, we have used ReLU activation function which takes maximum value between 0 and the input.

3.5 Backpropagation

For backpropagation, the previously mentioned weight initialization technique has helped to reduce repetition as neural network always tries to repeat the training process.

We tested with different combinations and got satisfactory result by dividing digital root of 9 for the first layer and digital root of 3 and 6 consecutively for each next pair of layers.

$$E_{h1} = \frac{W_1}{W_1 + W_2 + W_3} * E_O$$

$$E_{h2} = \frac{W_2}{W_1 + W_2 + W_3} * E_O$$

$$E_{h3} = \frac{W_3}{W_1 + W_2 + W_3} * E_O$$

Here, we can demonstrate it like this:

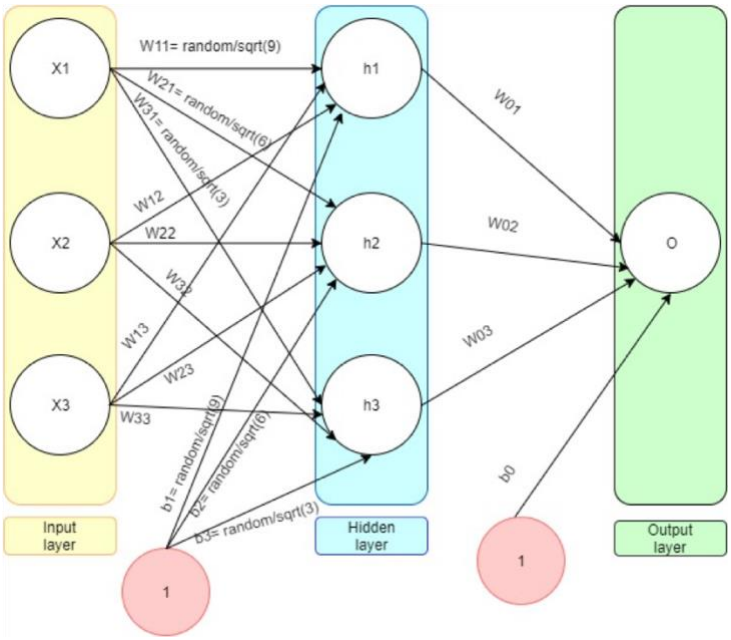


Fig 1: Custom backpropagation

In the figure above, we have three inputs X_1, X_2, X_3 and one output O.

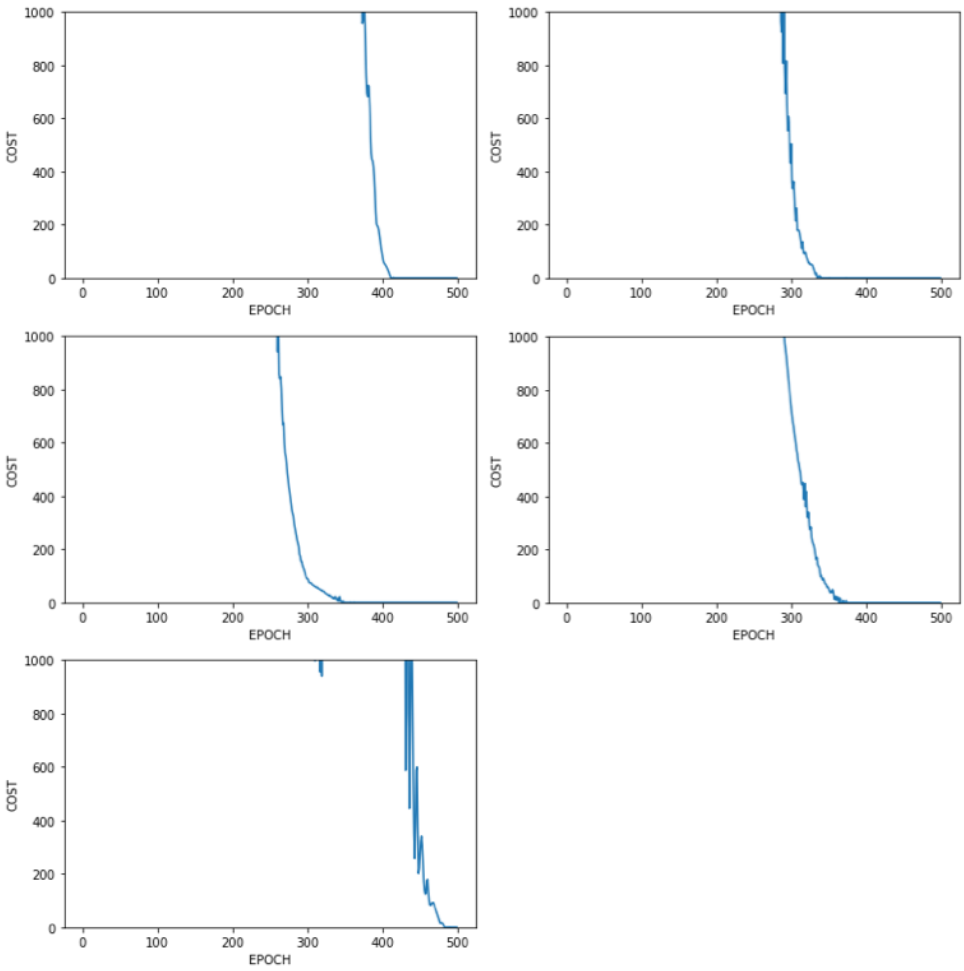
3.6 Model

To implement this model, we have used filter size 5*5 and number of filters used 32, 64, 128 consecutively. We have used TensorFlow API to implement this.

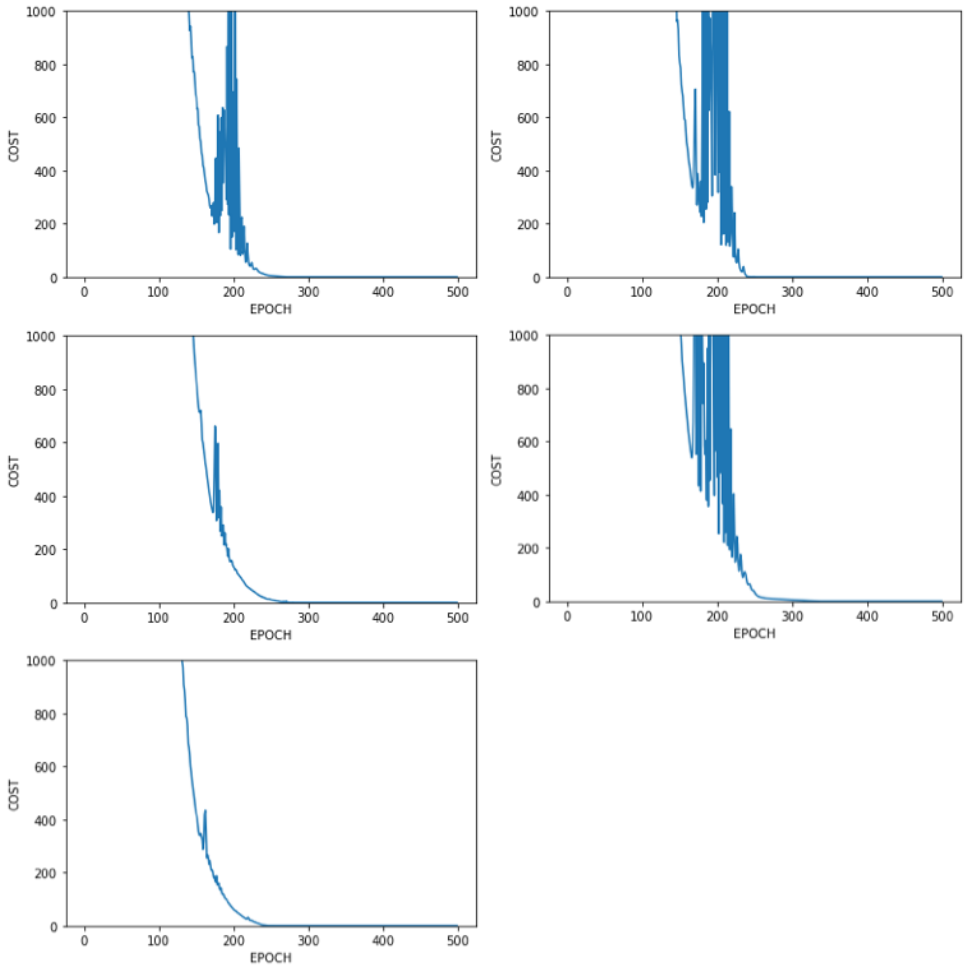
Chapter 4

Results or findings

4.1 Random weight initialization



4.2 9, 6, 3 Custom Weight Initialization



Chapter 5

Discussion

On convolutional neural network, more computational power is required in model training because of large number of epochs. We tried to reduce this large number of epochs. For this we have trailed with weight initialization and we got success message by reducing the number of epochs. We did number properties to finding pattern between numbers. We showed the way we can avoid this situation. Suddenly we find a pattern where 9,6,3 values are missing every time we double or half the digital root. In digital root initialization eliminate sqrt of 9,6,3 for faster convergence. In this initialization we show the result only for 5 times due to time shortage. In near future we will apply this digital root initialization at least 100 time on random variable.

Chapter 6

Conclusion

We proposed a new initialization architecture that makes the computation more efficient which was tested in several architectures. Avoiding the repetitive pattern of digital root of certain numbers, models were around 25-30% faster than before in our test cases. The objective for the next phase is to fine-tune the model for even more efficiency, as well as to test the initializer in many popular architectures and see what happens.

References

1. Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. IEEE International Conference on Computer Vision (ICCV 2015). 1502. 10.1109/ICCV.2015.123.

2. Glorot, Xavier & Bengio, Y.. (2010). Understanding the difficulty of training deep feedforward neural networks. Journal of Machine Learning Research - Proceedings Track. 9. 249-256.

3. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR, abs/1704.04861, 2017

4. Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. mobile networks for classification, detection and segmentation. CoRR, abs/1801.04381, 2018

5. Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv3: Searching for MobileNetV3. CoRR, abs/1905.02244, 2019

6. Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. CoRR, abs/1807.11626, 2018

7. Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.

8. Fan, Hongxiang & Liu, Shuanglong & Ferianc, Martin & Ng, Ho-Cheung & Que, Zhiqiang & Liu, Shen & Niu, Xinyu & Luk, Wayne. (2018). A Real-Time Object Detection Accelerator with Compressed SSDLite on FPGA. 14-21. 10.1109/FPT.2018.00014.

9. Agarap, Abien Fred. (2018). Deep Learning using Rectified Linear Units (ReLU).

10. Yang, Tien-Ju & Howard, Andrew & Chen, Bo & Zhang, Xiao & Go, Alec & Sze, Vivienne & Adam, Hartwig. (2018). NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications.

11. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition

12. Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He

13. I. Freeman, L. Roese-Koerner and A. Kummert, "EffNet: An Efficient Structure for Convolutional Neural Networks," 05 June 2018.

14. Andrew G. Howard, Menglong Zhu, Bo Chen and Dmitry Kalenichenko, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 17 April 2017.

15. X. Zhang, Xinyu Zhou, Mengxiao Lin and Jian Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," 07 Dec 2017.

16. S. Hochreiter and J. Schmidhuber, "Long short-term Memory," vol. 9, no. 8, p. 1735–1780, 1887.

17. Z. Andrew and S. Karen, "Very Deep Convolutional Networks for Large-Scale Image Recognition," University of Oxford, 10 April 2015.

18. H. Qassim, D. Feinzimer and A. Verma, "Compressed Residual-VGG16 CNN Model for Big Data Places," 10 01 2018.

19. Hussam Qassim, David Feinzimer, Abhishek Verma. Residual Squeeze VGG16

20. X. Z. S. R. a. J. S. K. He, "Deep Residual Learning for Image Recognition," 10 December 2015.

21. M. M. a. G. S. F. G. Larsson, "Ultra-deep neural networks without residuals," 24 May 2016.

22. Z. L. L. v. d. M. K. Q. W. Gao Huang, "Densely Connected Convolutional Networks," 25 August 2016.

23. K. G. J. S. Rupesh Kumar Srivastava, "Highway Networks," 03 May 2015.

24. https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

25. https://www.kaggle.com/ashishsaxena2209/animal-image-datasetdog-cat-and-panda

Thank You