CONTENTS

1. MOTIVATION

Deep Neural Network(DNN) has a lot of mathematical calculations anywhere in the range of 100‘s of thousand and up to billions of parameters in Advance Neural network. With advances in DNN, this is going to increase and so does power, memory and time requirements for them. Using these massive computations on edge devices, say a Mobile phone, Autonomous vehicle or other memory and battery sensitive devices have lot of constraint and this hinders the usage of amazing AI techniques for end user. As the majority of existing DNN encode weights and activation with 32-Bit full precision which need energy and memory that is out of reach for many embedded devices. Quantizing deep neural network increases throughput and bandwidth. However, this reduction in precision comes with a price which is the loss of accuracy. In the Quantized network, we lose some percentage of accuracy. This loss depends on the type of datasets, Mathematics used in Network and optimization done on the datasets. Theoretically, BNN needs 8 times less memory and it has 64 times speed increase compared to 8-Bit DNN if they can fully optimize the hardware. Keeping these gains in mind the main goal of this research project is to compare this trade-off in accuracy for three different cases- Full precision Network (FP-32), Quantized Network (16 bit or lower) and Binary Precision Network(1 Bit). Based on this evaluation this research project can make suggestions regarding the Feasibility of these quantized solutions on edge devices.

2. Related Work

AlexNet [KSH12] is the DNN which popularized DNN after winning ImageNet challenge. Later variant of this net design used to make improvement by making it more deeper in layer and achieve higher accuracy. Then Courbariaux et al. [CMH+16] worked on Binary Neural Network (BNN) and my work uses his approach coupled with modification done on his work to get better performance. One more API which is used in project is Larq API [GWB+19] an open-source deep-learning library for reduced precision neural net and it provides some amazing insight regarding model summaries like model size and the number of Multiply and Accumulate (MAC) operation compared to full precision Network. To explore the sparse network there is work done by Google team for imagenet classification and I have used their well developed neural net architecture for this research project. This project adopted a similar architecture as actual paper but explored their behaviour upon using them in reduced precision scale.

Main contribution of project is the selection of appropriate Neural network based on a literature survey and running some small experiments on various nets for selection and their evaluation. A major portion of work involves reducing precision(Binary and 8-Bit) of these networks and evaluate their result by running various experiments with optimization techniques developed by researchers, extraction and inferencing these results via graphs and tools like Tensorboard. The major portion of work in deep learning involves a series of experiment and without testing the models it is not easy to predict how a model will behave due to a large number of parameters involved. This is very time-consuming. Based on the correct inference of result we can develop a proper recommendation by leveraging past experiments or result.

3. INTRODUCTION

This research project report deals with the impact of reducing precision on the Deep Neural Network(DNN). The main focus was to evaluate how accuracy and model behave if precision points are reduced from the default way of 32 Bit floating point calculations to reduce precision as low as Binary Precision values. This project work performs a series of experiments and evaluations on two different kinds of neural network namely- AlexNet and GoogleNet version of InceptionNet. Each of them is evaluated on three different kinds of precision values- Full Precision (default), 8-Bit reduced precision and Binary precision.

After exploring motivation and related works this report is majorly divided into two parts. First section deals with theoretical background and second part explores implementation and results. Chapter 3 of report covers basics of quantization and a top-level overview of what different kinds of benefits quantization can bring to deep learning research is explained coupled with ways of quantization and looks around some mathematics for quantization. Then chapter 4 focuses on Binary and 8-Bit quantized Neural Network with layers and mathematical computation involves around this. Then it explains how binarization and quantization is achieved. The second part of this report(Chapter 5 and 6) deals with implementation aspects of three variations of two selected networks with final data and result interpretation. I have also covered top-level architecture details of both AlexNet and GoogleNet. In the end, report will look at some of the limitations of these results with suggestions for future work which can be based upon these.

3.1. QUANTIZATION

In deep neural network, Quantization is a process of approximating full precision floating-point numbers with lower precision low bit-width values. This, in turn, reduces memory requirement and mathematical calculation many folds. Deep Neural Network is experiment and innovation using mathematics and so quantization is techniques to use lower precision mathematics. It is a wider analogy term that covers various techniques to convert input high precision values from a large set to output values into a smaller precision value. There are different levels of quantization. We can have as high quantization as binary values (-1 and 1) from the 32-FP. One simple example is color image, if colour image is converted to gray scale it contains the same picture details minus the loss of pixel details(colour). But this immensely reduces the no of bits needed to store the image. So a trade-off between loss of some details and storage capacity is made. Since all numbers are represented by bits and so 8 bit have a range [- 128 to 127] compared to the range $[(2-2^{23}) \times 2^{127}, (2^{23}-2) \times 2^{127}]$ for FP32 so we can see how much difference is there for representation.

Quantization is used to speed up computation via reducing the precision of Weights, activation and power-hungry mathematical and matrix calculation. Which in turn reduce the need for memory footprint and energy usage. These constraints are particularly useful when we want to use artificial intelligence on end-user edge devices out of labs and university. Like mobile phones, automobiles system or vision-based robotics application area. Fig 3.1 shows how quantization impact the calculation compared to FP32 full precision. Some of the basic

Input Type	Accumulation Type	Relative math throughput	Bandwidth savings
FP16	FP16	8x	2x
INT8	INT32	16x	4x
INT4	INT32	32x	8x
INT1	INT32	128x	32x

Figure 3.1: Showing the impact of quantization on calculations relative to FP-32 precision [Wu19]

operations which is used as function and helper class in this project are explained below and they are straight out from basics of quantization. How values are quantized and scaled can be understood from below equation-

$$\mathbf{y_q} = round(s \cdot \text{clip}(\mathbf{y}, -\alpha, \alpha)) \tag{3.1}$$

where:

round = round to nearest value
α = clipping threshold
$$\text{clip}(x) = \begin{cases} -\alpha & , x \in (-\infty, -\alpha) \\ x & , x \in [-\alpha, \alpha) \\ \alpha & , x \in [\alpha, \infty] \end{cases}$$

Another point in quantization is use of symmetric range instead of asymmetric because this asymmetry introduces bias in model. For example, 8-Bit quantization is achieved as below-

Asymmetric range

$$[-128, 127], s = \frac{128}{\alpha} \tag{3.2}$$

Symmetric Range

$$[-127, 127], s = \frac{127}{\alpha} \tag{3.3}$$

Based on the above K-Bit quantization can be generalized in symmetric range as shown in equation 1.4

$$\left[-2^{k-1} - 1, 2^{k-1} - 1\right], s = \frac{2^{k-1} - 1}{\alpha} \tag{3.4}$$

where:

k = No. of bits for quantization

Clipping is a very important operation used in both quantization and binarization. This is used to get rid of outliers values which fall outside the range of quantization. Many researchers suggested that it is very difficult to quantize after training as this leads to a much bigger loss in accuracy, so train with quantization is always a better option. Quantization suffers from 2 major drawbacks- Non Differentiability: This is solved via the use of straight-through estimator (STE) and Non Continuous values: This is similar to digital and analog signals in digital electronics where in digital we are making an approximation of steps to replicate analog result values. This issue is solved by using two copies of weights, in DNN model fixed-point (reduced precision) is used for gradient calculation and floating-Point is used for gradient accumulation.

3.2. Methods of Quantization and Survey of Reduced Precision Networks

Usually when we talk about quantization we think in terms of 8-bit integer conversion from FP32. But other extremes of quantization like 4/3/2/1 bit is quite prevalent. The 1 Bit quantization, however, has a special place due to their extreme compression and limited state space to represent values. Another category of conversion from FP to INT value like 32-FP to 32-INT.

In short,some of the binary network and quantized network will be discussed. Main emphasis will be on BNN rather than 8-Bit network as they are very close to Full Precision. BNN network have a major issue with Stochastic Gradient Descent (SGD) due to binary value which is not the case for other higher bit quantization.

1. Binary Neural Network- Neural networks having binary weights and activations at run-time and when computing the parameters gradient at train-time [CMH+16]. Floating-point parameters are clipped between +1 and -1. This research project deals with this kind of network.
2. Ternary Weight Networks: Neural networks with weights constrained to +1, 0 and -1 [LZB+16]. Otherwise it is very close to BNN network.
3. XNOR Network: In this network both input and weights of the conv layers and Dense layer are approximated with the binary Value. Since conv layer need shift and dot product to finish their calculation, so author thought to convert dot product operation into binary one via approximation. Since in dot product we accumulate product between values, in a similar way XNOR operation multiply on Bit Level but accumulation is done via summation of all XNOR operation results. By use of binary encoding (-1 and -1 are Binary values than 0 and 1 are their Binary Encoding) and by leveraging popcount instruction in hardware we can count number of ones in binary value and multiply it by 2 and subtract the total No of bits that gives an integer value.

   This bit-wise operation is much more hardware efficient than a full precision and fixed precision powerful multiply and accumulate operation [RMO+16]. This network proved to be very popular and efficient. In compare to original BNN it uses an additional gain term to compensate the loss happen during binarization which increases the accuracy but the way of calculating gain in this Network is very costly. Here gain is calculated for every dot product in all conv layers then L1 norm of both activation and weight are

multiplied to find the gain value, which is costly.

4. Binary Weighted Network- In this network weights are constrained to Binary Values -1 and +1 which adds value to hardware because in this network many MAC operations are replaced by just accumulate operation and this brings a lot of saving due to removal of multiplication. Weights are binarized in both forward and backward propagation but during parameter update as shown in the algorithm we do not binarize and that is the reason SGD works well in this network.

   Algorithm of this network-
   Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η Ensure: updated parameters w_t and b_t 1. Forward propagation:

   - Forward propagation:
     $w_b \leftarrow$ binarize
     (w_{t-1}) For $k = 1$ to L, compute a_k knowing a_{k-1}, w_b and b_{t-1}
   - Backward propagation:
     Initialize output layer's activations gradient $\frac{\partial C}{\partial a_L}$
     For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and w_b
   - Parameter update:
     Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{d b_{t-1}}$ knowing $\frac{\partial C}{\partial a_k}$ and a_{k-1}
     $w_t \leftarrow \text{clip}\left(w_{t-1} - \eta \frac{\partial C}{\partial w_b}\right)$
     $b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$

   This network still uses 32-Bit full precision activation and parameters update are not binarized. Obtaining the best approximation of weights is the main goal of this network; which does not always translate into best approximation of final Network output. This may still lead to high accuracy loss [CBD15]. This was a precursor of BNN network and algorithm used in project is highly influenced form this.

5. Incremental Network Quantization(INQ): This is a lossless quantization technique to address high no of epoch requirement for good model accuracy. This method uses group-wise quantization, weight partitioning and retraining again and again. In INQ, first of all,weights are divided into different subgroups. Some groups of weights are quantized and the rest are used for re-training. In the second step, these low precise quantized weights is used to convert FP-32 values into low precision quantized value. In the third step, these low precision quantized weights are frozen and used for SGD to retrain the network and then the rest of the weights in the network are updated. Above three steps are repeated until all weights are quantized.

   In this network, weights are in the power of 2 as binary shift is more efficient for hardware but underlying computation in this network is still in FP. This model achieved model size compression but the speedup is not good [ZAY+17].

3.3. WHY QUANTIZATION OR BINARIZATION WORK

It is surprising to see quantization as low as binary one is working so fine as demonstrated by Deep learning community. One of the reasons is the value points of floating-point weights. Their distribution has a very narrow range and so even after quantization only some part of weight distribution is ignored. In the initial stages of quantization, there is serious problem with a drop in accuracy, which was later removed via a different kind of tweaking. By using methods like scaling floating-point range is mapped to their quantized range which helps us in scaling value to get better accuracy after quantization. Fig 3.2 shows how random values can be distributed when they are quantized. Here histogram on the right shows the actual distribution of weight values and left histogram shows their 4 Bit quantized counterparts. Here in quantized we record only some of the values accurately and rest are approximated or rounded off to the nearest values in the quantized range.

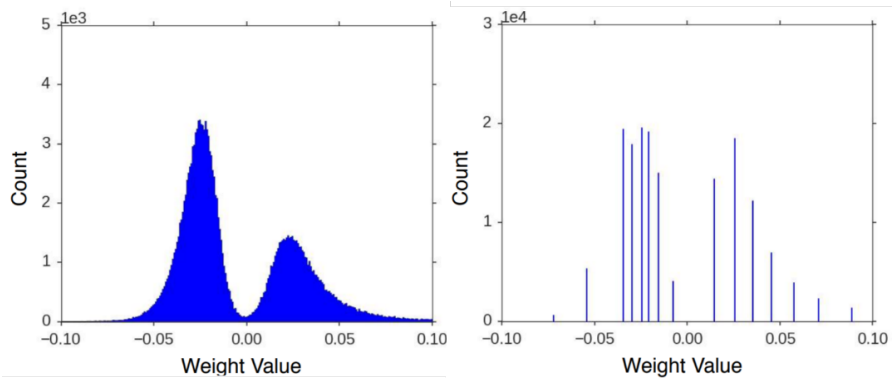


Figure 3.2: Diagram showing FP and 4 Bits weight distribution[MHJ16]

Since DNN have multiple parameters and over parameterized so in that case skipping some of the redundant information doesn't hurt the network and it works just fine. More details mathematics behind this can be found here [JKC+17].

4. Binary and 8-Bit Quantized Neural Network: Introduction and Implementation

Courbariaux et al. [CMH+16] developed and proposed the most famous BNN methodology on which many works were based. In BNN both weights and activation are binarized. Because this reduces memory footprint and complexity of mathematics.

- Weight Binarization: During forward pass weights are rounded to its nearest quantized or binarized value. So that very few parameters needs to be changed compared to full precision and this brings the problem of gradient descent which becomes zero in such rounding operation. In backpropagation, the gradient will be multiplied with zero and the previous layer will have zero learning. This is the place where loss is calculated for the network based on the quantized weight in the forward pass, but save the float weight and use them during backpropagation, they are called Latent weight. There are various ways to use this latent weight in backpropagation. Here in the project I have used STE(straight-through estimator) proposed by Courbariaux et al. [CMH+16].
- Activation Binarization: To use quantized or binarized input parameters in convolution layer activation functions(network uses the value provided after activation function for calculation) is replaced with their quantized or binarized counterpart before applying convolution operation. Courbariaux et al. [CMH+16] quantized activations to binary value -1 and +1 based on the sign of full precise activation value and via parameter clip. Values outside the range of [-1,+1] are discarded. In this project, I have used a binarized version of ***tanh*** activation for BNN and last layer activation function was Softmax. This is done to increase accuracy as the last layer output is very sensitive toward quantization.
- Hidden layer: Signed function non-linearity is used to get binary activation and weights are clipped during model train process. Weights are also quantized via straight-through estimator.
- Layers- Since the first layer of Neural network is receiving image it's always better to use the first layer with full precision. It helps in accuracy enhancement. Report will discuss the impact of statistics later in detail. The main reason is that inputs image have usually only three channels (red, green and blue for colour image) then their inner convolution layers structure, so the first layer convolution is less computationally intensive and they have less number of parameters.
- Rest all calculations and optimizations are done in the same way as regular deep neural network.

4.1. Algorithm of Binary Neural Network

The algorithm used in this project is same as proposed by Courbariaux et al.[CMH+16]. Same can be viewed in more details in section 1.3 as algorithm 1 in this cited research paper.

4.2. BINARIZATION MODULE AND ITS PROPERTY

In binarized and quantized neural network version layers are binarized and quantized and Values are quantized as well which are going to be fed to deep neural network. This is required to transform full precision input to quantized/binarized output and pseudo-gradient method in backward propagation. We are quantizing both input and kernels. These quantizing layers and quantizers are the main points which transform activations and weights of the corresponding layers.

Out of various available libraries available for deep learning, I have selected Keras due to its simple API layers which makes easier to implement the model and secondly, individual layers of Keras can be customised for BNN and quantization in an easier way. Key Parameters used to customize Keras layers to use them in BNN is as follows-

1. Quantizers- It is mathematically defined as -

$$x_q = q(x), \quad x_q \in \{-1, +1\}, x \in \mathbb{R} \tag{4.1}$$

So above mathematical function convert a wide range of full precision value to either +1 or -1 which is quantized output. Also, whole network needs to be quantized because even though inputs fed to network are binary but output values of deep layers is in form of integer due to reason that binary values are added in layers which in turn is no longer binary.

The exact quantization function used for binarization is as below-

$$q(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{4.2}$$

On the similar pattern gradient calculation is based on Straight-Through Estimator (STE) and here binarization is changed to clipped value in the backward propagation. In short, full precision weights are clipped to -1 and 1 using *weight_clip* as below-

$$\frac{\partial q(x)}{\partial x} = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases} \tag{4.3}$$

2. Binarized Layers - Layers of deep neural network needs to be binarized and to do so they need to understand how weights and activations to the layer are binarized. If Input_quantizer and Kernel_quantizer both are not passed through Straight Through estimator(STE) then this corresponds to Full Precision Network, If they are passed through Straight Through estimator(STE) then that means Binarized Neural Network and these values are clipped to binary values.

Mathematical equation employed at layers to find activation is as below-

$$\boldsymbol{y} = \sigma(f(q_{\text{kernel}}(\boldsymbol{w}), q_{\text{input}}(\boldsymbol{x})) + b) \tag{4.4}$$

Here Y is the output value based on $\boldsymbol{w}$ as weight, $\boldsymbol{x}$ input value, f as an operation which is multiplication in our case, σ as Activation function and $\boldsymbol{b}$ as Bias. Above computation can be visualized from Fig 4.1.

3. Quantized Activation: Keras and TensorFlow have their own rich library of Activation function but for BNN other customized version of Activation functions like- Signed Straight-Through Estimator (ste_sign) is used on the similar pattern as explained above for STE estimator. Another option is to use $\boldsymbol{binary\_tanh}$ which I have used in BNN.This activation act like the sign function in forward propagation however during back-propagation it act as below-

$$hard\_tanh(x) = 2 * \_hard\_sigmoid(x) - 1 \frac{\partial q(x)}{\partial x} = \begin{cases} 1 & |x| \leq 1 \\ 0 & |x| > 1 \end{cases} \tag{4.5}$$

So,gradient is cleared when |x| > 1 during backpropagation [CMH+16].

Fig 4.1 showcase how reduced precision network work from the top view. Keep in mind that BNN is an extreme version of quantization. So quantization in below diagram can be replaced by binarization for visual representation.

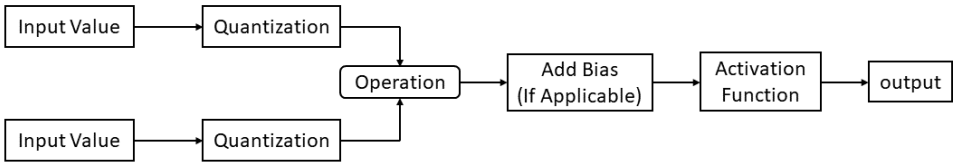


Figure 4.1: Diagram showing top-level Binary quantization operational Diagram on each layer

4.3. Implementation of Binary Neural Network

As discussed above, Keras API functionalities can be customized. One option is to binarize all layer's input to Binary value but from experiment, I found that by making the first layer and last layer as binary we are losing accuracy in the range of 1-1.5% for Kaggle Cats and Dogs image dataset. The number of mathematical operation and weights values are not so big compared to in between layers. So, input is fed in full precision to the network layer but Kernels were binarized and so does full precision weights clipped to +1 or -1. Similarly, at the last layer output, I have used full precision activation in the form of Softmax. The main reason to use Softmax is that it provides the probabilistic output and it makes the probabilistic sum of all possible prediction value as 1 and this property is helpful to extend project model to multi-class image dataset like MNIST(having 10 classes). So here my BNN model will say like with 0.6 probability I can say that image is Cat and with 0.4 probability I can say image as DOG; here sum of probability is 1. Also, the first and the last layer computation are a one-time operation and do not consume that much of resources. But from experiment, I found that if we quantize input value there is a reduction in 32 Bit MAC (Multiply and Accumulate) but losses are higher and model is less confident on prediction, so it depends on end-user what kind of trade-off is desired.

I ran some experiments with Batch Normalization after layers to improve efficiency and it allows each layer to learn independently and increase the stability of Neutral Network. It also normalizes the value after each layer, so it makes the transfer of learning and modelling it to different dataset easier from the model extension point of view[SSD+19].

Detailed implementation of BNN can be found in the code segment. I have used a model summary tool from Larq(https://larq.dev/)[GWB+19] to compute the memory footprint and reduced no of MAC operation in models combined with powerful Keras library to get an extensive model summary. This provided extensive insight regarding the impact of binarization on resources and other parameters and how they can reduce number of calculations on the neural network. There is a manifold difference in terms of memory footprint compared to full precision. During implementation statistics, report will discuss how much saving is there compared to 32 FP implementation for the corresponding model.

4.4. Implementation of 8-Bit Reduced Precision Neural Network

This is second set of implementation apart from full precision and binary Precision. This provided insight regarding what happen in case of moderate quantization(8-Bit) apart from full precision and extreme quantization(1-Bit). I have used 8 Bit quantization. This inherently enhances the range in which values can be stored which is 8 Bit now. This implementation is largely influenced by DoReFa-Net research paper [ZWY+18] and Minimum energy quantized neural Network [BGK+17]. In this quantized network weights and inputs both are trimmed down to 8-bit value instead of full precision and later same as BNN quantization is needed after computation step as well so that the final output is also quantized.

This quantized neural network can be treated as an extension of BNN; instead of Q=1-Bit, it will use Q=8 Bit. All weights and activations are quantized to Q bits as per below mathematical equation [BGK+17]-

$$q = \text{clip}\left(\frac{\text{round}\left(2^{Q-1} \times w\right)}{2^{Q-1}}, -1, 1 - 2^{-(Q-1)}\right) \tag{4.6}$$

For back-propagation Straight Through Estimator is used to find the derivative of the cost function with respect to Weight as below, where g_q is the estimator of $\frac{\partial C}{\partial q}$-

$$g_w / g_q = hardtanh(w) = clip(w, -1, 1) \tag{4.7}$$

Fig 4.2 shows this quantized network basic building block- Similar to BNN here weight is

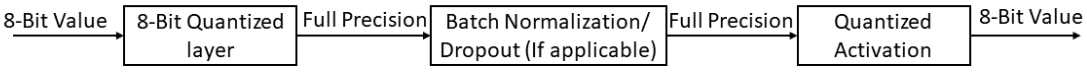


Figure 4.2: Quantized neural network basic building Block [BGK+17]

passed through clipper function which clip it in the interval [-1,1]. Work by Moons et.al

[BGK+17] showed that ***Quantized ReLU*** is better activation for the 4 and 8 Bit integer precision whereas ***quantized hardtanh*** is the better activation function for 1-Bit precision. All activations are clipped in the interval [-1,1] so that layer which is receiving these values get quantized input (in this project it is of type int-8).

4.5. Neural Network Selection

Selection of neural network needs a lot of literature research, code review and reading various neural Network developed so far. For this Research project, I have selected two networks- AlexNet and InceptionNet(GoogleNet). For both of them, 3 different kinds of models were implemented-

- Binarized Neural Network Version
- Quantized 8-Bit Neural Network Version
- Full Precision Neural Network Version

Reason of selection of these neural networks is based upon two parameters- simplicity and how they can be expanded. Alexnet is the main accelerator in the deep neural network and it is quite straightforward to estimate and good starting point to understand how binarization and quantization works and their impact can be studied in more details. Another point is no of parameters involved and so does memory requirement for storing and using deep neural network.

Another Network is InceptionNet and I have used its GoogleNet version. It is the latest and state of the Art. This network uses some of the latest developments in area of deep learning with the know-how of model size reduction coupled with accuracy improvement. Before this, there was thinking to go deeper which means increasing hidden layer but Inception net is more focused toward reducing computation and making model computation fast. This is the main motivation for me to select because the end goal of this project is to implement on Edge or end-user Hardware.

Another point is the model structure. Alexnet is sequential in nature however GoogleNet is Sparsely connected. So I thought it would be interesting to evaluate both kinds of a network to decide which architecture is better. Other possible networks were VGG net, Resnet and Le-Net. These three nets were state of the art themselves but I did not selected them due to the reason that they are very heavy on hardware and slow especially VGG net, even though performance is better.

Selecting order of filters and tuning them to get the best result is time consuming and difficult part of developing Neural network. On top of that, this decision needs to be made on every layer of the Neural Network. Here this inception modules comes into the picture. As shown in Fig 4.3 convolution with various dimensions are performed and then neural network selects best of them for calculation. But this network uses one more trick to reduce the number of dimensions and so does calculation involved. Instead of directly applying bigger filter multi-level filter is applied in convolution, which reduces no of calculations in a ma-

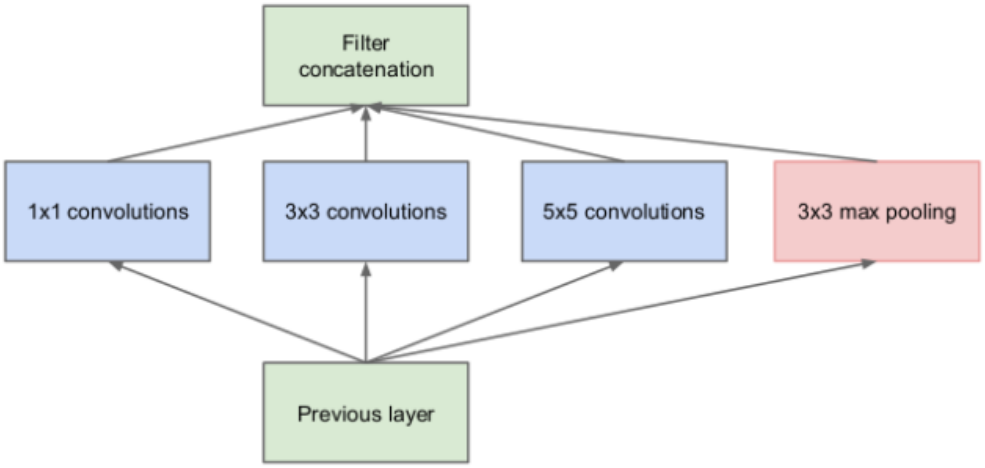


Figure 4.3: Simplified Version of Inception Net [SLJ+14]

trix.Same can be visualized in Fig 4.2 One observation found during model evaluation is that

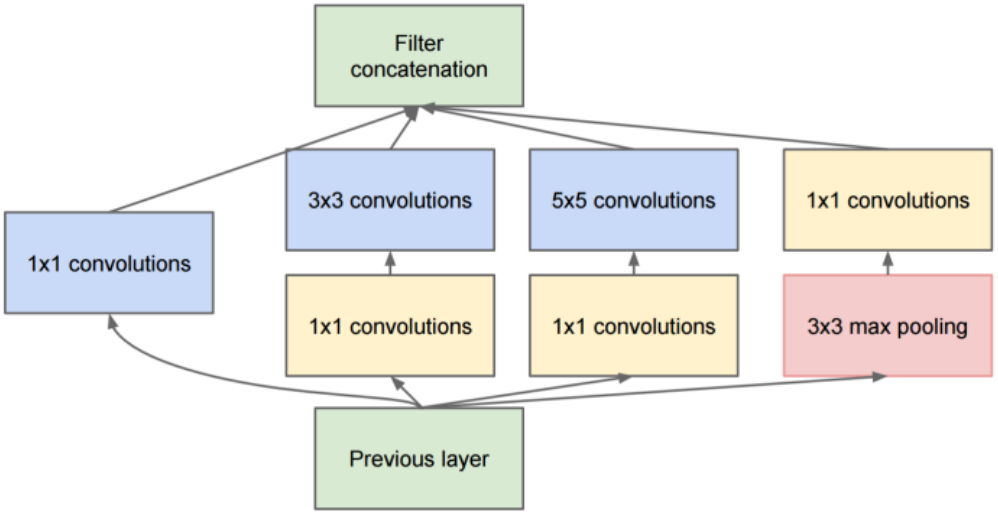


Figure 4.4: Reduced Dimension Version of Inception Net [SLJ+14]

AlexNet is very heavy on memory requirement. On using google Colaboratory server to run AlexNet model, server has to allocate around 3.5 to 4.5 GB of RAM depending upon optimization and other techniques, while GoogleNet reduced version (1 output architecture instead of 3 output) just needed 1.5 to 2 GB of RAM allocation on Google Colaboratory server. This itself shows us how these architecture differs from each other so this also influenced my network selection. Another point is no of parameters involved in the network. Please see below table No 4.1.

It can be seen here there is a drastic reduction in the number of parameters involved in

Neural Network	No. of parameters
Alexnet	143,233,410
Inception Net(Google-Net version)	5,975,602

Table 4.1: Number of parameters in Alexnet and Googlenet version used for evaluation

GoogleNet by 95.8%. Later when report discuss these network architecture in details then it can be seen that from model visual representation and number of layers involved it looks in the first instance that AlexNet is simple and it would have less number of parameters but upon their implementation for Cats and Dogs dataset it is opposite.

So it can be summarized that selection of neural network showcases both simplified and complex property of deep Neural Network and how they respond to quantization and binarization.

4.6. Framework and Toolboxes

I have used two very important tools available in the deep learning community- Keras and TensorFlow. They provide high-level API's, library and mathematical function with them. My research project leveraged these toolboxes library by customizing them for project Requirement. I also used API called Larq extensively in evaluating the BNN network.

4.7. Dataset and Preprocessing

I have used CATS and DOGS dataset provided by Kaggle and before using dataset fair amount of preprocessing is required before we can train and test neural network with these pictures. All images were converted into same size as per AlexNet and GoogleNet architecture requirements. The same size of image assists the model to learn in a better way and avoids extra padding. Good preprocessed dataset can significantly enhance the learning capability of model [YJS17]. Please note that converting images to Gray-scale (1 channel image) from the colour image(3 channel image) causes some loss in accuracy but they slightly speed up overall training and validation. I kept the image in colour scale for most of the experiments but some sets with gray-scale conversion were tested and it turned out that there is an additional loss in accuracy by doing this. All image sets are normalised before feeding it to Neural Network. One drawback of this dataset is fewer sample images to train and test. There is 8005 training images and 2023 test or validation images. But this dataset fastens the turnaround time for model testing.

4.8. MODEL TRAINING FOR ALEXNET AND GOOGLENET WITH PARAMETERS SELECTION

In this project “model.fit_generator()“ is used instead of “model.fit()“ functionality to train the network because using generator function dataset impact on memory is less, it is very easy to fit the data in memory compared to model.fit. So I used it from memory efficiency point of view and another reason is the extendibility of model to test the same network on large dataset which require very high memory. On top of that, data augmentation(increasing data diversity without the addition of new data point) functionality in model.fit_generator() is easy to use and save the data preprocessing effort.

5. Implementation of Full Precision and Reduced Precision AlexNet

In Appendix A, I have shown the actual Alexnet model used by me with all the in-between layers involved. This appendix coupled with code samples can provide quick insight into internal layers and representation.

5.1. Implementation of AlexNet on BNN

Same layers and architecture as discussed in original Alexnet paper [KSH12] is used, which means same types of kernels and filters as in the paper. The main point to note about implementation is that I have used below part of codes and supplied this as Kwargs to each layer so that operation on the layers are binarized-

```
kwargs = dict(input_quantizer="ste_sign",
              kernel_quantizer="ste_sign",
              kernel_constraint="weight_clip")
```

If the above values of kwargs are equal to None model will be Identical to full Precision or that value will not be binarized. Apart from this Keras layers are customized based on binary quantization as explained in section Binary Modules and property. Here I have used the last layer as "Softmax" activation instead of Binary Activation(*binary tanh*) and input is in full precision. Below is optimal result out of many conducted experiments by using various approaches. To get this result both training and validation accuracy with their losses were analyzed using Tensorboard to find the particular epoch number at which both training and validation accuracy are close to each other with reasonably high value of accuracy and simultaneously losses are low at the same point. This result is not maximum accuracy but the best possible value upon inspection which is comprised of epoch point having less loss and high accuracy,so model is neither overfitting or underfitting.

```
Best Training Accuracy:   48.44%
Best Validation Accuracy: 50.31%
Training loss for best Accuracy: 1.100
Validation loss for best Accuracy: 6.95
```

Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:   62.50%
Validation maximum accuracy: 50.31%
Training model minimum loss:   4.321
Validation model minimum loss: 1.133
```

Also, note that here we have not used Dropout feature and we ran our experiment for 30 epoch using EarlyStopping as callback. This callback checks model validation accuracy and

stops further epoch if an increase in the accuracy is not involved to avoid overfitting of the model.

Another point I observed during experiment that Loss parameter applied to the model during compilation has a very significant impact on the model accuracy and loss. Above result is obtained using categorical_crossentropy as loss parameter. I found that if Mean Squared Error(MSE) is used as loss parameter then training accuracy is almost the same as above, Validation accuracy suffer additional loss of approx 1% but on the other hand loss of model decreased significantly. During training, loss decreased by 10 times and during validation, it decreases by 2 to 3 times. So it does suggest that using MSE as loss our model is more confident in predicting correct value as suggested by the low loss but accuracy is low. Since main guiding factor was to find the best accuracy I compiled the result for categorical_crossentropy.

Impact of Dropout on BNN model: On experimenting with BNN model using Dropout of 0.5 after first fully connected layer I got below result:

```
Training maximum accuracy:   60.94%
Validation maximum accuracy: 48.12%
Training model minimum loss:   0.682
Validation model minimum loss: 0.693
```

As it can be seen here there is 2% additional loss of accuracy and a significant reduction in model loss. This low loss is expected as with the use of Dropout neural network becomes smarter by learning to identify internal structure independently without being sensitive to specific weights of neural network. As low loss can be analogous to the fact that Network is learning(like learning new concepts) not just mugging up from example.

Another fact that I found from experiment is the impact of binarizing last layer of activation and binarizing input. As believed, this resulted in Validation accuracy of approximately 45% which is almost 5% lower accuracy than non-binarized input and non-binarized last layer activation function. Additionally, this results in extra validation loss of 8 times compared to non-binarized first and last layer model. So it can deduced that it is better to avoid binarizing input and last layer which is neither good for accuracy nor model loss.

Larq tool API [GWB+19] allowed me to compare them instantly with 32-bit precision. Table 5.1 shows model summary of AlexNet on BNN. This table shows the benefits that BNN can

Total parameters	143 Million
Trainable params	143 Million
Non-trainable params	0
Actual Model size	17.14 MB
Float-32 Equivalent Model Size	546.39 MB
Total 32 Bit MAC operation	105,415,200
Total 1 Bit MAC operation	1,110,687,744

Table 5.1: Alexnet Binarised Model summary

potentially bring to deep learning community. The model size is an interesting part which

shows that full precision AlexNet is 32 times more in size compared to binary AlexNet. MAC operations are the most computationally expensive and the majority of them are tuned down to 1 bit compared to 32 bit which brings great saving in form of memory and power.

5.2. Implementation of Reduced precision (8 Bit) AlexNet

Here 8-Bit quantization is used for all layers like- Convolution and Dense. ***Glorat*** is used as Kernel initializer and then by using this initializer weight kernel is obtained which is later quantized to the 8-Bit range. Then convolution is performed and used in this 8 Bit quantized neural network. Next experiment was done on 8 Bit quantized value and similar to BNN version of Alexnet I have kept the first and last layer as Full precision. Activation function is quantized_relu for hidden layer. Model is compiled with categorical_crossentropy with Dropout of 0.5 after first fully connected layer. Epoch is 30 with EarlyStopping as callback similar to binary version of AlexNet. Similar to the binary version below I have compiled the optimal result of model based on observation of all losses and corresponding accuracy. This can be termed as the best compromise for Training and validation Accuracy and less loss-

```
Best Training Accuracy:   46.88%
Best Validation Accuracy: 53.22%
Training loss for best Accuracy: 9.93
Validation loss for best Accuracy: 4.74
```

Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:   67.19%
Validation maximum accuracy: 55.31%
Training model minimum loss:   3.217
Validation model minimum loss: 3.602
```

After analysing these statistics, model loss vs epoch graph and accuracy vs epoch graph, it can be visualized that for most of the epoch both validation loss and training loss is less in comparison to binary model loss. However, validation loss performance is much better and so does the validation accuracy. So it can be deduced that 8- Bit model is better in comparison to binary as validation accuracy statistics provides insight regarding how model performs on the test data or real images. So best accuracy is approx 3 percent higher compared to binary version. But further research is needed why Training accuracy and loss is not working in a better way as validation accuracy and loss. To find the explanation I have tested this model on larger Cats and Dogs dataset(20000+ images) and MNIST and I found that both 8-Bit model training and validation accuracy performs better than BNN model. So no of sample have a quite significant impact on model visualization.

To reduce the loss; model can be compiled using MSE loss and here I found a similar pattern as BNN version that loss decreased significantly with a sacrifice of 1-1.5% in accuracy.

However, the impact of 8-Bit quantization on the last layer which is output activation is very significant. In one experiment, I made last layer activation as quantized_relu opposed

to generally accepted softmax and I found that loss is undefined and makes very difficult to evaluate the model, which is very surprising and there is only one explanation for that and this is categorical_crossentropy as loss parameter. Because upon use of MSE as loss loss value was not undefined.By using MSE as loss and same quantized_relu as final layer activation it follow same pattern as Binary version- slight loss of accuracy with good improvement in loss parameter. This experiment also reaffirms that mathematics dictate the model performance and finding the right mathematics is the most innovative and difficult task for any dataset and network. As here just changing loss parameter can dramatically change the model perspective.

5.3. Implementation of Full precision AlexNet

This can be done in two ways. Firstly by extending binary Version by simply passing arguments as below and pass it to every layer in the form of kwargs-

```
kwargs = dict(input_quantizer=None,
              kernel_quantizer=None,
              kernel_constraint=None)
```

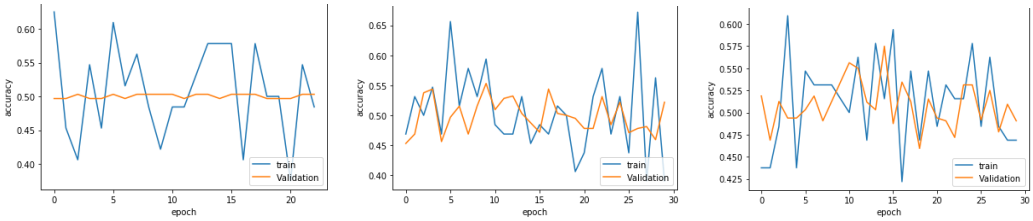
Another way is to use default Keras inbuilt layers and computation; as this is the default implementation in all deep learning libraries. I have chosen this approach because of the deep learning community likes this approach due to straightforwardness. Also, it is easy to implement without any overhead. Here last layer as Softmax activation is used and rest all layers were applied with Relu activation with no Dropout. Also, the model was compiled with loss as categorical_crossentropy in the same way as Binary and 8 Bit version. Below is the model output on using 30 Epoch and EarlyStopping as Callback. Similar to the binary version,optimal result of model is compiled based on observation of all losses and corresponding accuracy. This can be termed as best compromise for training and validation Accuracy and less loss.

```
Best Training Accuracy:   51.56%
Best Validation Accuracy: 57.50%
Training loss for best Accuracy:   0.7144
Validation loss for best Accuracy: 0.6842
```

Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:   60.94%
Validation maximum accuracy: 57.50%
Training model minimum loss:   0.68
Validation model minimum loss: 0.67
```

It can be seen here that FP-32 precision gives us best validation accuracy with the least amount of loss compare to others. One more observation I found that full precision model converges very quickly and initial loss was high but it goes down very quickly. Losses for the



(a) BNN model accuracy (b) 8-Bit model accuracy (c) FP-32 Bit Model Accuracy

Figure 5.1: AlexNet model accuracy graph for all three precision model

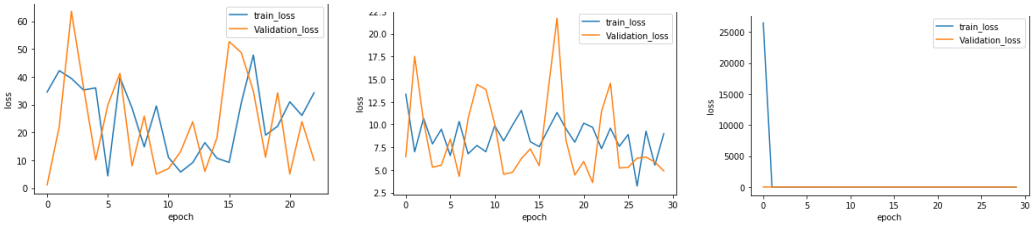
full precision model is 2 to 4 times lower than Binary and 8-Bit quantization value. This is very encouraging as low loss value provide high confidence for the model. This makes transfer learning (using this trained model on other dataset) more accurate and can be used for a broad range of applications. This model falls into the "Just Right" category due to the very close correlation between training and validation accuracy with very low loss value. Another pattern regarding less loss using Mean squared error (MSE) and Dropout were the same as observed for Binary and 8 Bit quantized network.

5.4. Discussion on AlexNet result for three variant of Network

Since loss is inversely proportional to the confidence of prediction value. Here Softmax is used in final output layer,so more loss means even though the model is predicting labels correctly but it is not much confident in that prediction. This also suggests that if model is transferred to the real world then more losses means there are more chances of wrong labelling. So it can be concluded that in order of certainty FP is greater than 8-Bit quantized network and which in turn greater than binary network of AlexNet. The same pattern is observed for validation or test accuracy, where FP has the highest accuracy followed by 8-Bit Network and then followed by binary network as expected. All three network properties is summarized in Table 5.2. Fig 5.1 and 5.2 shows Accuracy and loss for all three models respectively across epochs.

AlexNet Complete Summary				
Network Version of AlexNet	Training Optimal Accuracy	Validation Optimal Accuracy	Training loss for Optimal accuracy	Validation loss for Optimal accuracy
Binary Precision	48.44%	50.31%	1.100	6.95
8-Bit Precision	46.88%	53.22%	9.93	4.74
Full Precision	51.56%	57.50%	0.7144	0.6842

Table 5.2: Summary of AlexNet Optimal accuracy and loss on Binary, 8-Bit and Full Precision model



(a) BNN Model Loss (b) 8-Bit model loss (c) FP-32 bit model loss

Figure 5.2: AlexNet model loss graph for all three precision model

So as expected full precision model is best. But from experiment, it is observed that if we tweak the binary or quantized network by using techniques like Dropout, Regularizer, Batch Normalization and selecting the correct type of loss we can close this gap and predict model with much confidence. From the graph visualization it is observed that loss and accuracy graph for each Epoch fluctuate very highly for binary and 8-bit network compared to full precision, this makes the selection of best epoch very hard from a user point of view. This can be attributed to less no of data samples as discussed earlier. Lesser the data more sensitive the network to quantization. K-Fold cross-validation can be used to improve these models. The k-fold cross-validation estimator has less variance than a single hold-out set estimator, which is very crucial for small dataset. Impact and additional statistics can be collected for these techniques in future work. This will be interesting to explore as if this works fine, we can train the model on dataset having less samples and use them for real-world application, transfer learning.

5.5. Additional implementation of AlexNet on MNIST dataset

To test the impact of quantization on more diverse and bigger dataset this experiment is conducted on MNIST which has 60000 train image and 10000 test images scattered across 10 different classes.Upon conducting experiment with 30 epoch and same layer architecture as used for Cats and Dogs dataset, It is observed that even for quantized version loss is low and fluctuations as seen in earlier experiment is not present in this case. One more observation was improvement in accuracy with increase in number of epoch was observed throughout unlike previous experiments. This experiment re-affirm earlier stated fact that less sample dataset was are more prone to losses, which in turn causes problem in transfer learning for model trained with those data. I did one more experiment on MNIST dataset with BNN model but network architecture was different from AlexNet, only 3 hidden layers with 5 epoch were used and Validation accuracy reached around 97%. This accuracy is very close to some of the best DNN model in FP which gives accuracy in range of 99.5%.

One more experiment on 25000 test images of Cats and Dogs showed same pattern as MNIST where with increase in epoch there is gradual increase in Accuracy and decrease in loss for all precision unlike high variation observed across each epoch in earlier dataset of Cats and Dogs. However, FP model converged very quickly compared to other precision.

6. Implementation of Full Precision and Reduced Precision GoogleNet

GoogleNet is a flavour of Inception Net, to utilize computing resources inside the network, based on increasing the depth and the width of the network at the same time keeping the computational budget constraint [SLJ+14]. This network is trying to move from fully connected to a sparsely connected structure. This is a network inside a network which is trying to find and estimate an optimal local sparse structure. In contrast to the fully connected dense sequential model like AlexNet; the sparse network does not connect to every node. This is the reason why GoogleNet which look so a big architecture but saves computational parameters many times. In fully connected network having back to back convolution layer in chained fashion, then any increase in the number of filters for the convolution layer increase the computation quadratically. So sparsely connected network address this computational bottleneck by introducing sparse network on the overall network and also inside the convolutions. I have implemented a reduced number of output layers compared to actual GoogleNet

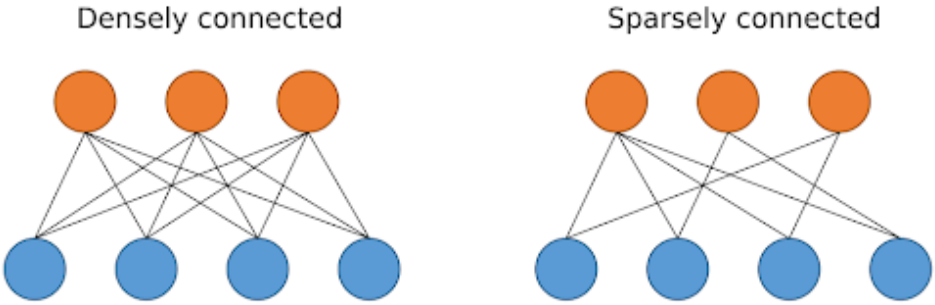


Figure 6.1: Diagram showing the difference in the fully connected dense network and sparse Network [Ala18]

because of very high computational demand of this network. Authors of GoogleNet suggested that training took about a week with high-end GPU to converge for Imagenet dataset. There is only one change from actual GoogleNet architecture and that is the number of the auxiliary output port. In sequential Network, there is only one output but original GoogleNet have 3 output so they are called auxiliary output. I worked with only one output. The exact architecture of my implementation is shown in Appendix B. This Network is 22 layer deep and looks so huge but the power lies in the fact that it still has much lesser parameter than AlexNet. In the next section, report will discuss all three precision(Binary, 8-Bit and Full precision) implementations of GoogleNet. All compiled results shown below are based on “EarlyStopping“ as callback and number of epoch selected is 150. The loss for all listed values by default is “categorical_crossentropy“ and “Adam“ as an optimizer unless otherwise specified.

Basic mathematics related to binary layers, activation and other neural network customization is the same as explained in the earlier chapters. Same is applicable for callbacks used to optimize network like early stopping. The main change from previous AlexNet network is the architecture which is used here as explained in GoogleNet research paper [SLJ+14]. Here I have used the first layer on the input side Activation as *"Relu"* instead of *"Binary tanh"* and so does the last dense layer as Softmax instead of binararized Activation value. However, for the first layer, I have used signed Straight-Through Estimator(ste_sign) as a way of quantizing Kernel and weight is clipped. Due to less number of parameters, it allowed me to use epoch as high as 150 to test with the use of early checking as a callback to avoid overfitting. Similar to the AlexNet BNN version the optimal result of model is compiled based on observation of all losses and corresponding accuracy. This can be termed as the best compromise for training and validation accuracy and less loss.

```
Best Training Accuracy:     51.56%
Best Validation Accuracy:   51.56%
Training loss for best Accuracy:   0.8481
Validation loss for best Accuracy: 0.6934
```

Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:   67.19%
Validation maximum accuracy: 55.62%
Training model minimum loss:   0.64
Validation model minimum loss: 0.69
```

As it can be seen that here that model is successful in reducing the loss significantly and accuracy at the same point compared to AlexNet. Best Validation accuracy increased by 1% and maximum accuracy increased by 5%. However, the main gain is obtained in model loss, where it is observed that the best training loss is now 30% lower and the best validation loss is 90% lower than their AlexNet counterpart. Another point by exploring the best training and validation is that the model falls under just fit category as both accuracies are the same and this makes the model more interesting to study due to no over or under-fitting.

This network already had Dropout implemented in architecture. I ran some tests on this model by making first layer activation as *"Relu"* instead of *"binary_tanh"* and it is observed that loss becomes higher and accuracy on both training and validation dataset is reduced. This is similar to earlier experiments on AlexNet network which showed that binarizing first and the last layer impact network in terms of accuracy and loss metrics.

Now coming to an interesting comparison of this binary version with full precision version using a model summary function in larq API [GWB+19].

Table 6.1 shows us that binarized model uses 94.87% less memory than their Full precision counterparts.

Total parameters	5.98 Million
Trainable parameters	5.98 Million
Non-trainable parameters	0
Actual Model size	1.17 MB
Float-32 Equivalent Model Size	22.80 MB
Total 32 Bit MAC operation	494,889,728
Total 1 Bit MAC operation	1,254,041,856

Table 6.1: GoogleNet binarized model summary

I ran some experiments on Binary GoogleNet with Mean squared error(MSE) loss and found that by using MSE on an average loss is reduced by 50% over the epoch and accuracy is comparable to the model used above with categorical_crossentropy as a loss. But I am going to keep the main result compiled with categorical_crossentropy to ease out the process of comparing with AlexNet model which showed better performance with categorical_crossentropy in terms of accuracy.

6.2. Implementation of Reduced precision (8 Bit) GoogleNet

In the same way as AlexNet 8-Bit quantized network uses the same mathematics, callbacks and customization in this GoogleNet version. As earlier, first and the last layer are not quantized. Except for layers and activation which are tuned as per 8-Bit Quantization. Basic network architecture is in line with actual GoogleNet paper [SLJ+14]. The last layer is softmax activation and the first layer is Relu. All other in between layers are using a quantized version of relu "quantized_relu" similar to the AlexNet binary version I have compiled the optimal result of model based on observation of all losses and corresponding accuracy. This can be termed as the best compromise for Training and validation Accuracy and less loss.

```
Best Training Accuracy:   54.69 %
Best Validation Accuracy: 53.44 %
Training loss for best Accuracy:   0.7434
Validation loss for best Accuracy: 0.7876
```

Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:     60.94 %
Validation maximum accuracy:   55.31 %
Training model minimum loss:   0.67
Validation model minimum loss: 0.688
```

As expected there is improvement in the accuracy for both Training and validation data due to additional precision values compared to binary version of GoogleNet in range of 2-3% for both. However, the loss for both is in the same range and comparable. In comparison to

AlexNet version of 8-Bit, the loss is low and Accuracy is high for both train and validation dataset with less model size and parameters.

On using loss as MSE it is found that over the range of epoch loss is decreased by 50% on an average and decrease of 1-2% in accuracy across the range of epoch.

6.3. Implementation of Full Precision GoogleNet

I have used Keras inbuilt libraries for full precision implementation. Here Softmax as Activation is used in last layer and rest all layers were applied with Relu activation with Dropout value as defined in GoogleNet paper. Also, the model was compiled with loss as categorical_crossentropy in the same way as Binary and 8 Bit version. Below is the model output on using 150 Epoch and "EarlyStopping" as Callback. Similar to the earlier version, below optimal result is compiled for model based on observation of all losses and corresponding accuracy. This can be termed as the best compromise for Training and validation Accuracy and less loss.

```
Best Training Accuracy:   56.25 %
Best Validation Accuracy: 54.92 %
Training loss for best Accuracy:   0.6977
Validation loss for best Accuracy: 0.6892
```

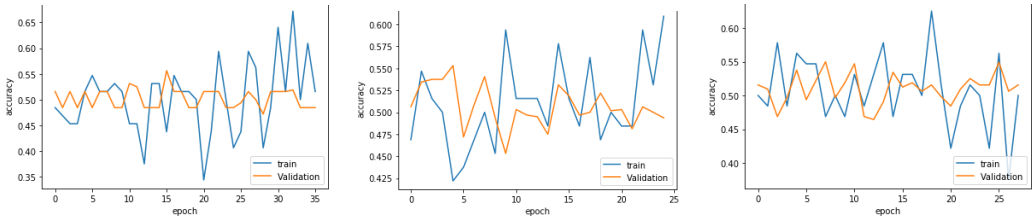
Top statistics for accuracy and loss for this model-

```
Training maximum accuracy:     62.50 %
Validation maximum accuracy:   55.00 %
Training model minimum loss:   0.67
Validation model minimum loss: 0.67
```

As it can be seen here that FP-32 precision produces best validation accuracy with the least amount of loss compared to all others. Here initial loss was very high(loss=18) but it goes down very quickly as the number of epoch progresses and then settles down. Unlike AlexNet where losses in reduced precision is quite higher compared to FP, GoogleNet losses does not show such high degree of difference in reduced precision compared to FP model.This is a good observation that GoogleNet seems to be working as a better fit network for reduced precision than AlexNet not only on metrics of loss but also for accuracy. Also, the less lossy model makes transfer learning using GoogleNet more accurate not only for full precision but for reduced precision as well and it will have more usability. Another pattern regarding less loss using Mean squared error (MSE) were the same as observed for Binary and 8 Bit quantized network where we have further decrease in loss and a slight reduction in accuracy.

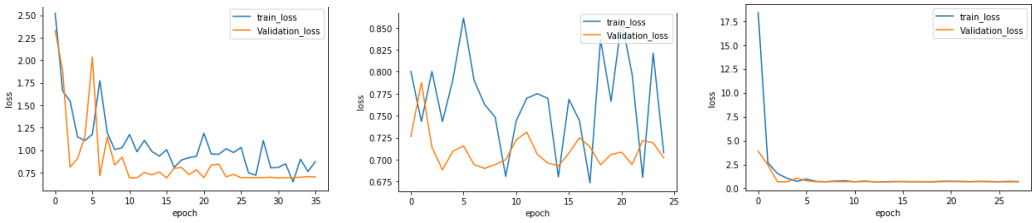
6.4. Discussion on GoogleNet result for three variant of Network

After implementing all three version of GoogleNet on the same dataset. The best accuracy is summarized as shown in table 6.2. Fig 6.2 and 6.3 shows accuracy and loss of three different



(a) BNN model accuracy (b) 8-Bit model accuracy (c) FP-32 Bit model accuracy

Figure 6.2: GoogleNet Model Accuracy Graph for all three precision model



(a) BNN model loss (b) 8-Bit Model Loss (c) FP-32 Bit model loss

Figure 6.3: GoogleNet model loss graph for all three precision model

precision of GoogleNet across each epoch respectively.

GoogleNet Complete Summary				
Network Version of GoogleNet	Training Optimal Accuracy	Validation Optimal Accuracy	Training loss for Optimal accuracy	Validation loss for Optimal accuracy
Binary Precision	51.56 %	51.56 %	0.8481	0.6934
8-Bit Precision	54.69 %	53.44 %	0.7434	0.7876
Full Precision	56.25 %	54.92 %	0.6977	0.6892

Table 6.2: Comparison of accuracy and loss of GoogleNet model variants

It can be seen that this model loss is very low and accuracy is also high in comparison to AlexNet. Binary GoogleNet validation accuracy increased by 1.25%, 8-Bit version accuracy rose by 2% but full precision accuracy has a reduction of 2.5% in validation accuracy but a 5% increase in training accuracy. This is a strange result as GoogleNet is supposed to be more accurate than AlexNet. This can be attributed to our small dataset, unlike imagenet datasets. To get concrete analysis for full precision on various kinds of dataset we need to explore further with a varying range of big datasets like Imagenet and other real-world dataset like wildfire and earthquakes to get concrete evidence on the similar way as I did for AlexnNet using MNIST and larger cats and dogs dataset.

These comprehensive results suggest that approximating sparse structure network can im-

prove the result and at the same time keep the computational budget low. It can be a very efficient technique to use if explored and optimized further. This type of DNN can be used on the end-user edge devices which requires less computation and demand memory and power efficiency.

7. Discussion,Limitations and Future Work

In this project, I could not work on large datasets like CIFAR and ImageNet. To work on these datasets powerful GPUs or TPU with a fair amount of RAM capacity is needed. Due to the large volume of dataset epochs to train are quite large. For the evaluation purpose, small datasets were used so this is one of the limitations of this work. Another point of concern can be a type of optimization and tricks used in the neural network. It is quite possible that tricks and techniques used for full precision can be detrimental for reduced precision network or network architecture issue like batch normalization can be very tricky to use. Where and how to use batch normalization proved to be very crucial. A Recent study by Sabri et.al [SBM+19] they found the main role of BatchNorm is to avoid exploding gradients in the case of BNNs. This finding suggests that the common initialization methods developed for full-precision networks are irrelevant to BNNs [SBM+19]. However, I have used here the same type of layers for full and reduced precision to get difference in accuracy for same kind of layers in DNN. I tested some of DNN techniques like dropout, loss type change and layer activation function but many more techniques need to be tested to get greater picture.

I have used software implementation of DNN without considering hardware aspects and how they will work out for this software implementation. To understand this bottleneck let us say a model is trained and optimized for 16 Bit precision but evaluating this model operation on our laptop which are usually optimized for 32-Bit FP, will not show any gain from reduced precision model as our laptop hardware will not support 16 Bit calculation due to hardware limitations and keep on using FP-32. However microprocessor architecture like Hashwell and new Intel CPU started to support Floating Point MAC per clock per core. So without optimized hardware and their proper support, it will be hard to justify the gain shown in software implementation. This hardware limitation also restrict us to predict actual hardware speedup and power efficiency from a theoretical point of view. One might conclude that the bit-wise operation is 32 times faster than FP-32. But, execution time and number of instructions can not be mapped so easily. Each instruction has its own no. of clock cycles to execute. In modern CPU and GPU scheduling algorithm is dynamic and the number of cycles to generate instructions is based on on other instructions that have been received previously. CPUs and GPUs are optimized for different kinds of instructions.

7.1. Enhancing and Improving BNN

The biggest limitation of the BNN network is their overhead of saving full precision weight for gradient descent in back-propagation. This property is going to stay unless researchers find a way to get a backpropagation with binary precision value and non zero gradients at the same time.

This work can be extended further with varied nature of datasets and by running various experiments on BNN network by keeping in mind that BNN and reduced precision can not be treated in the same way as FP network. These are different domain and requires new thinking. One such example is to use new kind of optimizer like Binary Optimizer (BOP) [HWG+19] which tries to address the issues of latent weight. Another future work can be

dedicated to finding optimal loss and compilation techniques which can increase the accuracy. This is established from this project work that how by changing loss type from categorical_crossentropy to Mean squared error reduces loss by a good amount. So further losses types and other optimizers like SGD, BOP etc can be explored instead of Adam optimizer used in this project. GoogleNet is used here in their actual form(This work might not be optimized for Binary or quantized network) so it can be explored with another optimizers and other innovative way in the same line as discussed above; as this kind of sparse network can open a way for less computationally intensive DNN with very good accuracy. Some other techniques can be-

- Partial Binarization: This approach suggests to evaluate the model first and then binarize only some layers which brings most amount of compression in model and leaving the most important layers at full precision.TaiJiNet [THW17] divide Kernel into two segments, some are binarized and others are kept as it is. Some approach divide layers in full precision and binarization.
- Learning Rate: Higher learning rate increases the training rate speed as in general BNN takes extra time compared to Non BNN version.
- Padding: Usual padding approach is to use zero padding but this introduces third term in BNN apart from +1 and -1 which caused incompatibility in bit-wise operation because in network there is no encoding for digit 0, we have already used 0 and 1 for +1 and -1. This brings problem to use BNN on ASICS and FPGA. So if we can pad with +1 or -1 then its better.
- Scaling with the gain term: In binarization more care is taken regarding the sign of the inputs and their magnitude is ignored so next layer have no idea about magnitude. So if we can leaf out ideas from XNOR net and improve it to find gain term in less computation it can be a big addition for BNN accuracy.

8. CONCLUSION

I have presented reduced precision network and demonstrated that Binary and 8-Bit Quantized network does not perform so bad in comparison to their FP counterparts; considering their possible benefits like less memory and power consumption. I did this through comparison after evaluating results and inferring those details. One more conclusion is benefits which sparsely connected network brings for us which have better accuracy and less memory and possibly less power requirement. This justifies selection of one sequential network and one sparsely connected network. This work showed that further research in Binary sparsely connected network can pave out the use of these highly efficient networks on mobile, robotics and embedded application. This work showed that on small dataset having less training images we lost around 3.5% accuracy compared to power-hungry full precision model. So we are closer to implement quantized and binary network on small devices with a small loss in accuracy. Note that using additional layers in BNN and Quantized network there is overhead of calculation as additional calculation on top of inbuilt Keras layers is performed.

During course of project finding correct no of epoch with no over or underfittng was challenge, which was solved by using callback, problems like where to use batchnorm and dropout and their impact on DNN can be solved only by experimenting. On the other hand deducing fact and answering unusual behaviour need alternate set of experiment and coming up with new set of experiment and fitting proper mathematics was most time consuming. Like explaining behaviour of high loss and accuracy fluctuation in small dataset is achieved via experiment on multi-class dataset like MNIST. Still I could not complete other possible experiments due to time constraint and further work by exploring parameters can make these model more robust and transferable in reduced precision space. Using code of this project reduced precision network can be trained and then it can be used to evaluate real time CNN/image classification task either with full precision or with reduced precision during evaluation. However training process is much more computationally intensive compared to evaluation.

I did experiments with some of the available and known sets of activation function, Batch Normalization and specific number of layers to conclude that even if same high precision layers are just converted to Reduced precision without further optimization, it is still producing very good result. On top of that if we could optimize these we can get more comparable result to the Full precision. In previous sections, I have detailed how this work can be carried forward with experimenting new set of compilation, activation and hidden layer parameters. As in MNIST dataset I got 96% accuracy with just 3 Hidden layer using Binary Network(not exactly an AlexNet model but other random design with three layers). All these evaluations and theory discussed point toward advantage and gains, which is very high in reduced precision and they should be more researched for discovering new optimized model having sole focus on reduced and binary precision.

9. ACKNOWLEDGEMENTS

I would like to thank Dipl.Ing Thomas Horn and Dipl.-Inf Matthias Sauppe from Professorship Circuit and System Design at Technische Universität Chemnitz for their support and guidance throughout this research project. Without their support, it would not be possible to finish this project on time. I am also thankful to Dipl.-Ing. Jingrui Yu from Professorship Digital- und Schaltungstechnik for his help on some of the implementation aspects.

REFERENCES

[Ala18] ALAVI, Amir: Using a VNN architecture. In: *Using a VNN architecture* (2018), Jul. https://amiralavi.net/blog/2018/07/29/vnn-implementation

[BGK+17] BERT ; GOETSCHALCKX ; KOEN ; BERCKELAER, Van ; NICK ; MARIAN: Minimum Energy Quantized Neural Networks. In: *arXiv.org* (2017), Nov. https://arxiv.org/abs/1711.00215

[CBD15] COURBARIAUX, Matthieu ; BENGIO, Yoshua ; DAVID, Jean-Pierre: BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In: *CoRR* abs/1511.00363 (2015). http://arxiv.org/abs/1511.00363

[CMH+16] COURBARIAUX ; MATTHIEU ; HUBARA ; ITAY ; BENGIO ; YOSHUA: Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to 1 or -1. In: *arXiv.org* (2016), Mar. https://arxiv.org/abs/1602.02830

[GWB+19] GEIGER, Lukas ; WIDDICOMBE, James ; BAKHTIARI, Arash ; HELWEGEN, Koen ; HEUSS, Maria ; NUSSELDER, Roeland: *Larq: An Open-Source Deep Learning Library for Training Binarized Neural Networks*. Web page. https://larq.dev. Version: 2019

[HWG+19] HELWEGEN, Koen ; WIDDICOMBE, James ; GEIGER, Lukas ; LIU, Zechun ; CHENG, Kwang-Ting ; NUSSELDER, Roeland: Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization. In: *CoRR* abs/1906.02107 (2019). http://arxiv.org/abs/1906.02107

[JKC+17] JACOB, Benoit ; KLIGYS, Skirmantas ; CHEN, Bo ; ZHU, Menglong ; TANG, Matthew ; HOWARD, Andrew G. ; ADAM, Hartwig ; KALENICHENKO, Dmitry: Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In: *CoRR* abs/1712.05877 (2017). http://arxiv.org/abs/1712.05877

[KSH12] KRIZHEVSKY, Alex ; SUTSKEVER, Ilya ; HINTON, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. Version: 2012. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. In: PEREIRA, F. (Hrsg.) ; BURGES, C. J. C. (Hrsg.) ; BOTTOU, L. (Hrsg.) ; WEINBERGER, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, 1097–1105

[LZB+16] LI ; ZHANG ; BO ; LIU ; BIN: Ternary Weight Networks. In: *arXiv.org* (2016), Nov. https://arxiv.org/abs/1605.04711

[MHJ16] MAO ; HUIZI ; J., William: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In: *arXiv.org* (2016), Feb. https://arxiv.org/abs/1510.00149

[RMO+16] RASTEGARI ; MOHAMMAD ; ORDONEZ ; VICENTE ; REDMON ; JOSEPH ; FARHADI ; ALI: XNOR-Net: ImageNet Classification Using Binary Convolutional Neural

Networks. In: *arXiv.org* (2016), Aug. https://arxiv.org/abs/1603.05279

[SBM+19] SARI ; BELBAHRI ; MOULOUD ; NIA ; PARTOVI, Vahid: How Does Batch Normalization Help Binary Training? In: *arXiv.org* (2019), Oct. https://arxiv.org/abs/1909.09139

[SLJ+14] SZEGEDY, Christian ; LIU, Wei ; JIA, Yangqing ; SERMANET, Pierre ; REED, Scott E. ; ANGUELOV, Dragomir ; ERHAN, Dumitru ; VANHOUCKE, Vincent ; RABINOVICH, Andrew: Going Deeper with Convolutions. In: *CoRR* abs/1409.4842 (2014). http://arxiv.org/abs/1409.4842

[SSD+19] SANTURKAR ; SHIBANI ; DIMITRIS ; ILYAS ; ANDREW ; MADRY ; ALEKSANDER: How Does Batch Normalization Help Optimization? In: *arXiv.org* (2019), Apr. https://arxiv.org/abs/1805.11604

[THW17] TANG, Wei ; HUA, Gang ; WANG, Liang: *How to Train a Compact Binary Neural Network with High Accuracy?: Semantic Scholar*. https://www.semanticscholar.org/paper/How-to-Train-a-Compact-Binary-Neural-Network-with-Tang-Hua/75dd30fde950bc8996b5105fff6f9fc3c1d2dbae. Version: Feb 2017

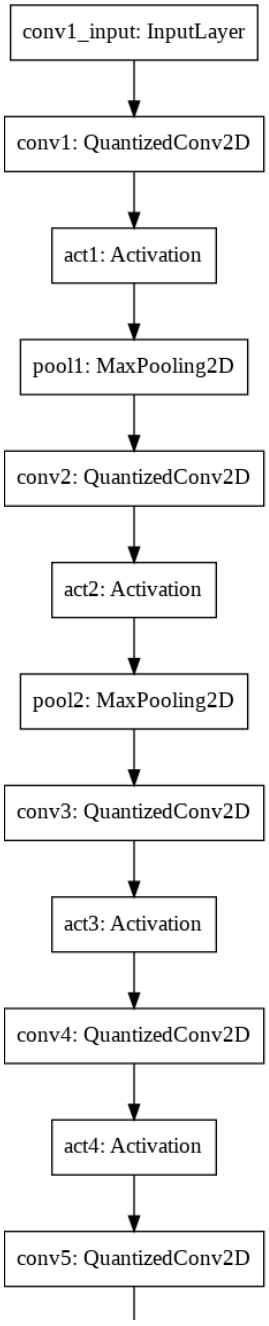
[Wu19] WU, Hao: *Low Precision Inference on GPU*. https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf. Version: 2019

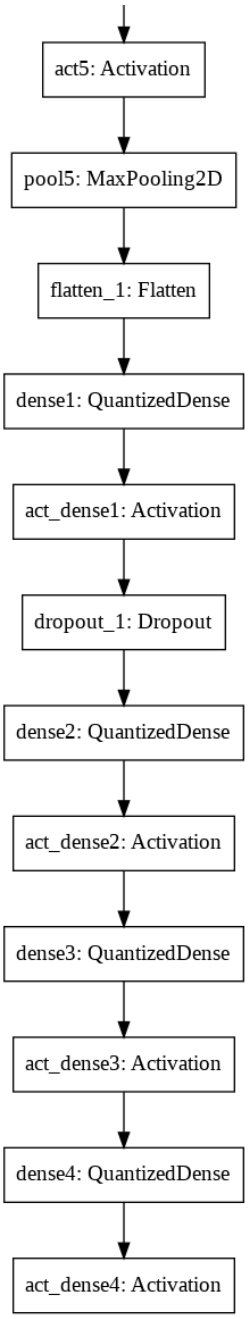
[YJS17] YIM ; JONGHWA ; SOHN: Enhancing the Performance of Convolutional Neural Networks on Quality Degraded Datasets. In: *arXiv.org* (2017), Oct. https://arxiv.org/abs/1710.06805

[ZAY+17] ZHOU ; AOJUN ; YAO ; GUO ; XU ; LIN ; CHEN ; YURONG: Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. In: *arXiv.org* (2017), Aug. https://arxiv.org/abs/1702.03044

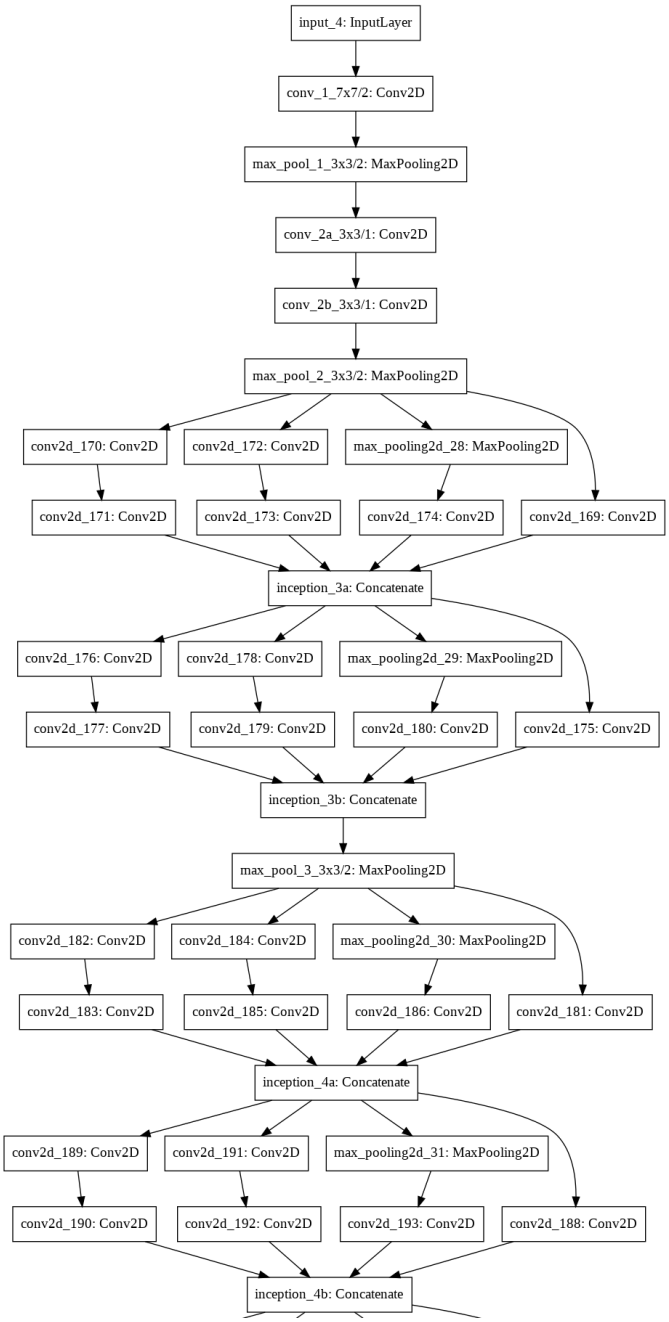
[ZWY+18] ZHOU ; WU ; YUXIN ; ZEKUN ; ZHOU ; XINYU ; WEN ; ZOU ; YUHENG: DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. In: *arXiv.org* (2018), Feb. https://arxiv.org/abs/1606.06160

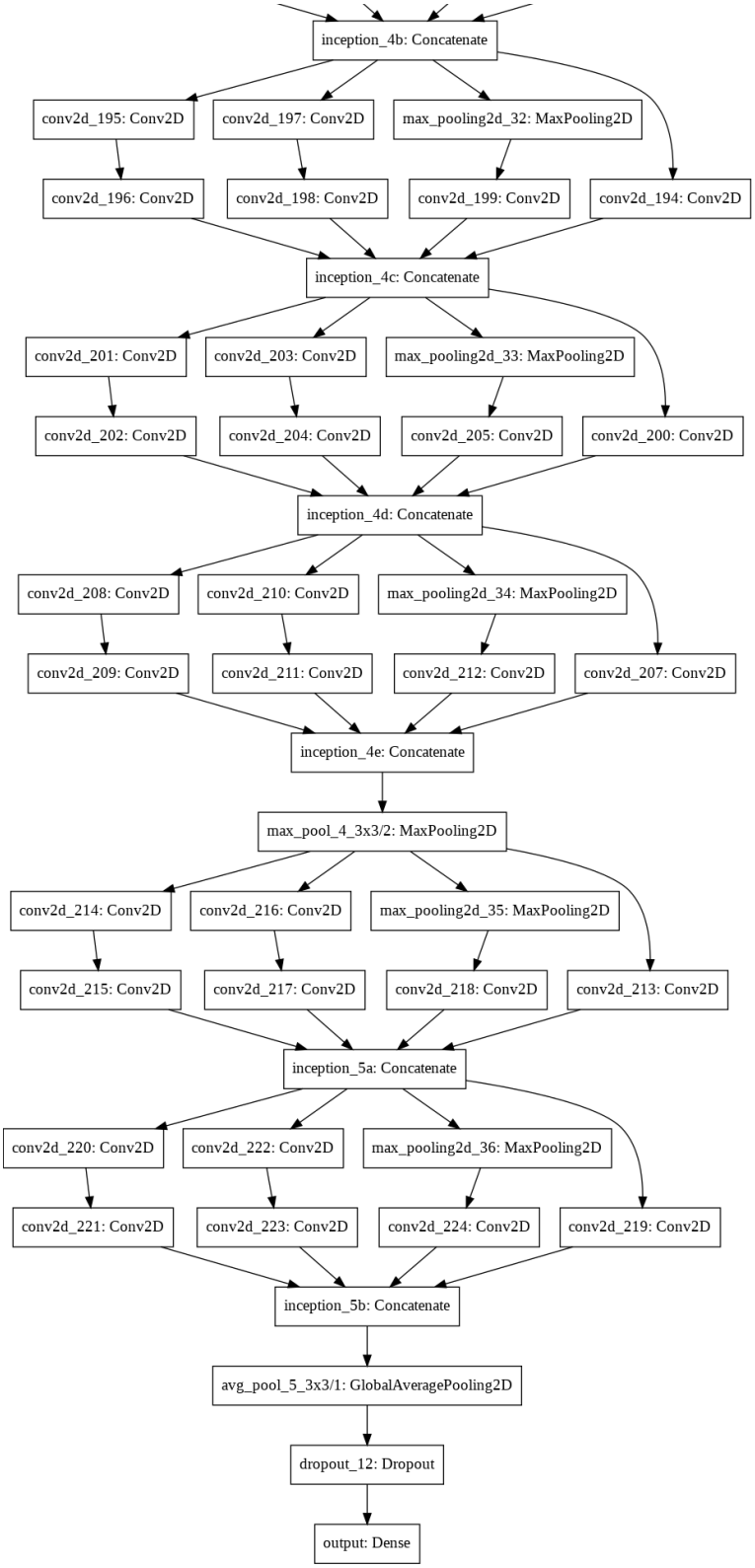
A. AlexNet Architecture Diagram




act5: Activation
pool5: MaxPooling2D
flatten_1: Flatten
dense1: QuantizedDense
act_dense1: Activation
dropout_1: Dropout
dense2: QuantizedDense
act_dense2: Activation
dense3: QuantizedDense
act_dense3: Activation
dense4: QuantizedDense
act_dense4: Activation

B. GoogleNet Architecture Diagram




inception_4b: Concatenate
conv2d_195: Conv2D
conv2d_197: Conv2D
max_pooling2d_32: MaxPooling2D
conv2d_196: Conv2D
conv2d_198: Conv2D
conv2d_199: Conv2D
conv2d_194: Conv2D
inception_4c: Concatenate
conv2d_201: Conv2D
conv2d_203: Conv2D
max_pooling2d_33: MaxPooling2D
conv2d_202: Conv2D
conv2d_204: Conv2D
conv2d_205: Conv2D
conv2d_200: Conv2D
inception_4d: Concatenate
conv2d_208: Conv2D
conv2d_210: Conv2D
max_pooling2d_34: MaxPooling2D
conv2d_209: Conv2D
conv2d_211: Conv2D
conv2d_212: Conv2D
conv2d_207: Conv2D
inception_4e: Concatenate
max_pool_4_3x3/2: MaxPooling2D
conv2d_214: Conv2D
conv2d_216: Conv2D
max_pooling2d_35: MaxPooling2D
conv2d_215: Conv2D
conv2d_217: Conv2D
conv2d_218: Conv2D
conv2d_213: Conv2D
inception_5a: Concatenate
conv2d_220: Conv2D
conv2d_222: Conv2D
max_pooling2d_36: MaxPooling2D
conv2d_221: Conv2D
conv2d_223: Conv2D
conv2d_224: Conv2D
conv2d_219: Conv2D
inception_5b: Concatenate
avg_pool_5_3x3/1: GlobalAveragePooling2D
dropout_12: Dropout
output: Dense