POLYTECH NANTES
École d'ingénieurs de l'université de Nantes

POLYTECH NANTES
Département Electronique et Technologies Numériques
5ème Année

Comparison and compression of neural networks on Cifar10 Dataset

Projet Technique Report

Supervisors: Suiyi Ling and Patrick Le Callet

Abstract: This paper details and presents experiments done in the field of neural network compression. A model selection is first done. The final objective being to run the compressed network on an edge device.

SALMON Martin - FINGE Omar
February 2020

Contents

1. Introduction 4
2. Related Work 5
2.1. First Steps 5
2.2. Lightweight Models 5
2.3. Compression 6
2.4. Neural Network on Android Smartphones 6
3. Finding the best compressed model for embedded systems 7
4. Methods and models used 7
4.1. Three main Models 7
4.1.1. AlexNet 8
4.1.2. SimpleNet 9
4.1.3. SimpleNetSlim 10
4.2. Optimization 11
4.2.1. Optimizer 11
4.2.2. Batch Normalisation 13
4.2.3. Dropout 15
4.2.4. L2 Regularisation 15
4.3. Model Compression 15
4.3.1. Pruning 15
4.3.2. 8-bit Quantisation 16
5. Experimental Results 17
5.1. Setup 17
5.1.1. Dataset Used 17
5.1.2. Toolbox 18
5.2. Method results 18
5.2.1. Dropout 18
5.2.2. L2 Regularisation 19
5.2.3. Pruning 20
5.2.4. 8-bit quantisation 22
5.3. Model Comparison 23

5.4. Weights and Flops 26
6. Conclusion 27
7. Future Work 28
8. References 29

Table of Figures

Figure 1 - Architecture of AlexNet 8
Figure 2 - Architecture of SimpleNet 9
Figure 3 - Architecture of SimpleNetSlim 10
Figure 4 – Comparison of the different Optimizers 12
Figure 5 - The effect of Batch Normalisation 14
Figure 6 - Pruning 16
Figure 7 – Dropout Effect 19
Figure 8 - L2 Regularisation Effect 20
Figure 9 - Pruning Effect on validation accuracy with 90% sparsity 21
Figure 10 - Accuracy Comparison 23
Figure 11 - Precision Comparison 24
Figure 12 - SimpleNet Confusion Matrix 25
Figure 13 - SimpleNetSlim Confusion Matrix 25
Figure 14 - AlexNet Confusion Matrix 26

Table 1 - Pruning effect on size with 95% sparsity 21
Table 2 - Pruning Effect on size with 90% sparsity 22
Table 3 - Final Accuracy values 24
Table 4 – Final Precision values 24

1. Introduction

Lately, the interest in neural networks has been growing quickly. Many deep learning techniques allow for significant results on a lot of datasets, in multiple fields, such as object detection, semantic segmentation or classification. However, the need to put deep neural networks on smaller devices such as phones or other embedded devices pushed the world to do more research to reduce the size and the computation time of efficient neural networks with minimal effect on performance. Various compression-related works have been done and are available online.

The need of accurate neural networks has led to develop popular methods. According to [1], CNN is one of the best performing methods because of its deep architecture. The deeper it is, the better the results are. The main problem CNN faces is size. An efficient CNN often equals a large network thus hardware limitations become a true issue.

In this paper, we will first explain how to reproduce benchmark neural networks with TensorFlow and Python, and why we chose them. Before studying and facing compression, an efficient neural network is first needed. A proved efficient network makes accurate comparison possible and helps us make the right choices and adjustments in order to get the best result. The SimpleNet architecture [2] has been proved to outperform other architectures using a technique of reducing the CNN depth empirically. Then, we'll speak about the different available and easy-to-implement compression techniques that we've done, like pruning and quantization. Finally, we'll show that it is important to estimate the weights and flops the model is using, because not all devices are able to run such networks, and consumption mustn't be too high if it needs to run on a device with an average battery.

2. Related Work

Being both beginners in Machine Learning, we've surfed the Internet looking for papers explaining Neural Networks and ways to implement them in order to learn and gain experience. In this section, we will cite the most important papers we read and detail them.

2.1. First Steps

In order to implement more complex models, we first needed to understand and create a simple neural network model. When that's done, we can easily change the dataset and the architecture as we please. To achieve this, we used the keras-master which implements ResNet and VGG architectures with showed results.

Many datasets are available online and are easy to import with python and keras librairies. Since many models were developed on famous datasets, we chose to use them in our project, especially Cifar10. We followed w

2.2. Lightweight Models

Since CNN is usually linked with size issues, one approach is to try running lighter models. These could allow for use on mobile and embedded devices with average system specifications.

One of the models used in our project is called "SimpleNet". This neural network has been created and improved empirically (experience and intuition). The architecture proposed is a simple CNN with 13 layers and a few filters for each layer. This model has been proved to be efficient and outperforms most of the neural network for the Cifar10/Cifar100 and fashion MNIST Datasets.

2.3. Compression

Various techniques of compression have been developed to aim for smaller models. These methods have proved to be very efficient with minimal effect on performance. Methods such as parameter pruning by reducing redundant parameters or low rank factorization using matrix decomposition are very developed and documented online. [3]

We can also combine compression techniques for even better results. That's what DeepSZ did [4] to obtain spectacular results during their tests.

2.4. Neural Network on Android Smartphones

Since the goal of this project is to run the neural network on an embedded device, a benchmark on the current available SoCs (System On Chip) is very important in order to test the energy consumption, the battery power and the computational power needed to run that Neural Network.

One paper online [5] covers this and details the pros and cons of every SoC. It was written in 2018 and the website linked [6] (table 2, page 11) states that for 2018, the best SoC is the HiSilicon Kirin 970 on the Huawei P20 Pro.

3. Finding the best compressed model for embedded systems

In this project, we wanted to find the best possible model to go on embedded systems. Meaning that we're trying to get the best accuracy from different models to compare them on the same dataset while having the minimum number of FLOPs. Keeping an eye on hyper-parameters is very important, so different tests are done to get the best parameters for one model. Once a model is well stated, their accuracy, size, weights or even flops can be compared with others. Precision and accuracy can also help us asses the comparison of 2 models fairly well.

Once a model is selected, different techniques of compression can be applied, these will be detailed later in this paper. Of course, compression methods must be applied to different models to see if one can be more compressed than another, without losing more.

4. Methods and models used

4.1. Three main Models

Due to the time provided for this project, the number of models used for experiments have to be limited. That is why only 3 models are implemented through which we want to show their main differences in terms of size and performance.

4.1.1. AlexNet

AlexNet is a model created in 2013 by Alex Krizhevsky and was a revolution in the field of convolutional neural networks since it broke a record in visual recognition accuracy because of the possibility of depth with convolutional layers [7]. Even though the model is large, good results are now feasible thanks to the recent evolution in performance for GPUs.

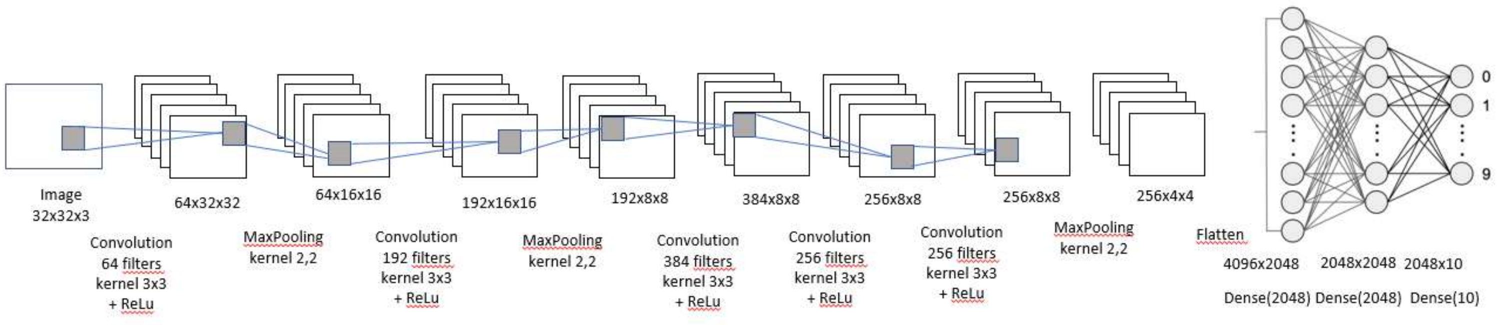


Image 32x32x3
64x32x32
Convolution 64 filters kernel 3x3 + ReLu
64x16x16
MaxPooling kernel 2,2
192x16x16
Convolution 192 filters kernel 3x3 + ReLu
192x8x8
MaxPooling kernel 2,2
384x8x8
Convolution 384 filters kernel 3x3 + ReLu
256x8x8
Convolution 256 filters kernel 3x3 + ReLu
256x8x8
Convolution 256 filters kernel 3x3 + ReLu
256x4x4
MaxPooling kernel 2,2
Flatten
4096x2048
2048x2048
2048x10
Dense(2048) Dense(2048) Dense(10)
0
1
9

Figure 1 - Architecture of AlexNet

The model adapted for the CIFAR-10 dataset is made of 5 3x3 kernel convolutional layers accompanied by a ReLu Activation for each and 3 2x2 kernel max pooling layers after the first, the second and the fifth convolutional layer. The specificity of its architecture remains in its depth of convolutional layers. As described in the figure 1, 64 filters are used in the first convolutional layer, then 192 and 384 in the second and third one. Three Dense layers are finally added to get the final 10 classes. Due to these deep layers, the model's size is significant, containing 172,032 neurons and 14,859,082 parameters, thus needing a powerful GPU to run for many epochs. Methods to get parameters and more will be detailed in part 5.3 Model Compression. Tests of AlexNet's accuracy on CIFAR-10 available online show a mean of 90% accuracy, which we tried to reproduce.

Even if AlexNet has proved its efficiency over the recent years, it has been outperformed by some new models inspired by it, implemented with new techniques. Furthermore, as it will be explained later in this paper, a smaller model is needed for an embedded device.

4.1.2. SimpleNet

SimpleNet has been created in 2016. Its use of the CNN architecture in a simple and empirical way, making a lightweight model, permitted to outperform deeper and heavier architectures such as ResNet, VGG or even AlexNet. [2]. When testing on big datasets such as CIFAR-100, SimpleNet has proved to be very efficient, offering good performance results and keeping the model light. To match the needs of our project, where a light model is needed to be run with low computational power, SimpleNet seemed like a wise choice. By default, Dropout and Batch Normalization are added to handle overfitting, but some testing, detailed in the optimization part later on, is done to understand the difference.

After adapting the model to the CIFAR-10 dataset, it has 13 convolutional layers interleaved with 5 max Pooling layers. Using 3x3 kernels for convolution provides more efficiency than having a bigger kernel because of the precision of small kernels. Having bigger kernels could induce a loss of information as explained in the SimpleNet Paper [2] page 7.

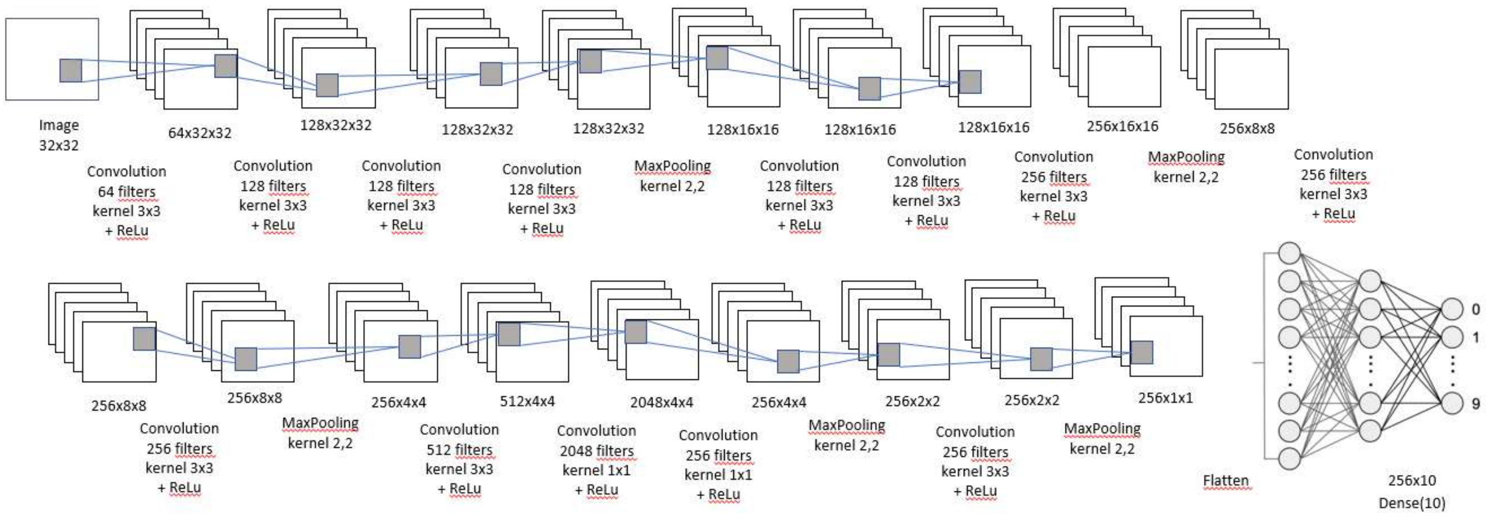


Image 32x32
64x32x32
128x32x32
128x32x32
128x32x32
128x16x16
128x16x16
128x16x16
256x16x16
256x8x8
Convolution 64 filters kernel 3x3 + ReLu
Convolution 128 filters kernel 3x3 + ReLu
Convolution 128 filters kernel 3x3 + ReLu
Convolution 128 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 128 filters kernel 3x3 + ReLu
Convolution 128 filters kernel 3x3 + ReLu
Convolution 256 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 256 filters kernel 3x3 + ReLu
256x8x8
256x8x8
256x4x4
512x4x4
2048x4x4
256x4x4
256x2x2
256x2x2
256x1x1
Convolution 256 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 512 filters kernel 3x3 + ReLu
Convolution 2048 filters kernel 1x1 + ReLu
Convolution 256 filters kernel 1x1 + ReLu
MaxPooling kernel 2,2
Convolution 256 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Flatten
0
1
9
256x10 Dense(10)

Figure 2 - Architecture of SimpleNet

As many other models, this one is created with small convolutional layers that gradually increase in size as the model goes deeper. Only one dense layer is needed to provide the final result on the 10 classes. Even with that many layers and having more neurons (668,672), the model is still much smaller in size than AlexNet: Only 5,497,226 Parameters, namely, 3 times less parameters.

Compared to AlexNet, its accuracy can reach around 94%. The calculation time needed to train the model for more than 100 epochs is approximatively the same as AlexNet.

4.1.3. SimpleNetSlim

SimpleNetSlim is relatively the same model as SimpleNet. It has been developed to satisfy the size and computational power issues. Actually, it's only a modified version of SimpleNet since it has exactly the same layers, except that the number of filters is decreased to get less weights, flops, a smaller size and a faster training. Therefore, training a neural network such as SimpleNetSlim on embedded device is even more feasible than any other one. Those characteristics are studied in the 5. Model Compression part.

The main challenge of this model is to get as much accuracy as possible with the smallest possible model. Thus, a compromise has to be done between the size of the model, and accuracy achieved with.

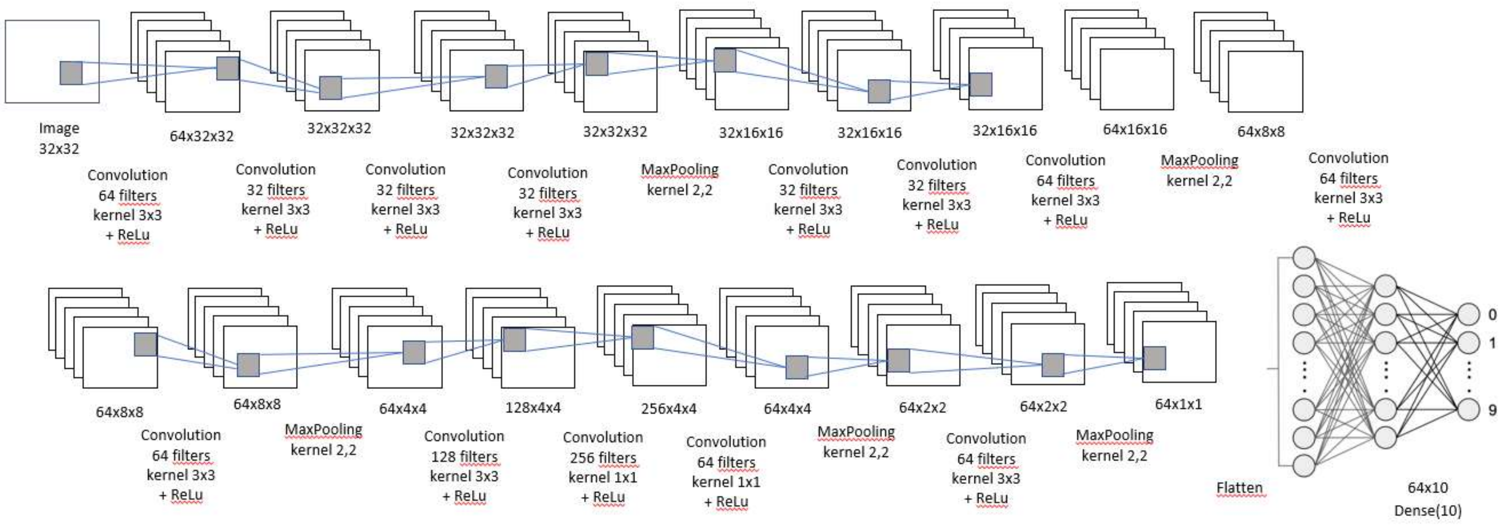


Image 32x32
64x32x32
32x32x32
32x32x32
32x32x32
32x16x16
32x16x16
32x16x16
64x16x16
64x8x8
Convolution 64 filters kernel 3x3 + ReLu
Convolution 32 filters kernel 3x3 + ReLu
Convolution 32 filters kernel 3x3 + ReLu
Convolution 32 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 32 filters kernel 3x3 + ReLu
Convolution 32 filters kernel 3x3 + ReLu
Convolution 64 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 64 filters kernel 3x3 + ReLu
64x8x8
64x8x8
64x4x4
128x4x4
256x4x4
64x4x4
64x2x2
64x2x2
64x1x1
Convolution 64 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Convolution 128 filters kernel 3x3 + ReLu
Convolution 256 filters kernel 1x1 + ReLu
Convolution 64 filters kernel 1x1 + ReLu
MaxPooling kernel 2,2
Convolution 64 filters kernel 3x3 + ReLu
MaxPooling kernel 2,2
Flatten
0
1
9
64x10 Dense(10)

Figure 3 - Architecture of SimpleNetSlim

As seen in the last part, this is exactly the SimpleNet Model, with changes on the number of filters. According to tests online, this model can achieve 92%~ accuracy [8]. Its number of neurons is 212.224 which is slightly more than AlexNet and much lower than SimpleNet but its parameters are decreased from 5,497,226 on SimpleNet to 312.682.

All 3 models are now set up to be trained and tested and as we can see in their description, they are different in terms of number of parameters, but also in terms of structure, especially between AlexNet with less layers, and

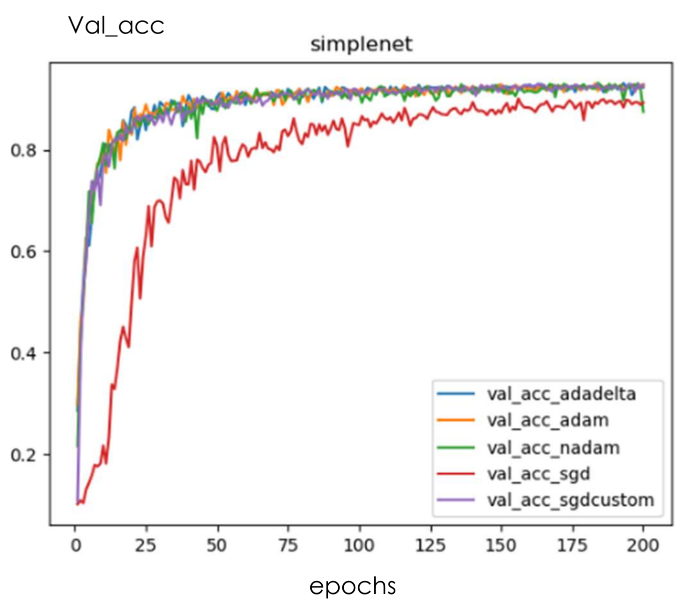
SimpleNet/SimpleNetSlim with deeper and lighter convolutional layers. This allows to evaluate now which one could the best fit for an embedded device where the accuracy needs at its best with minimal size.

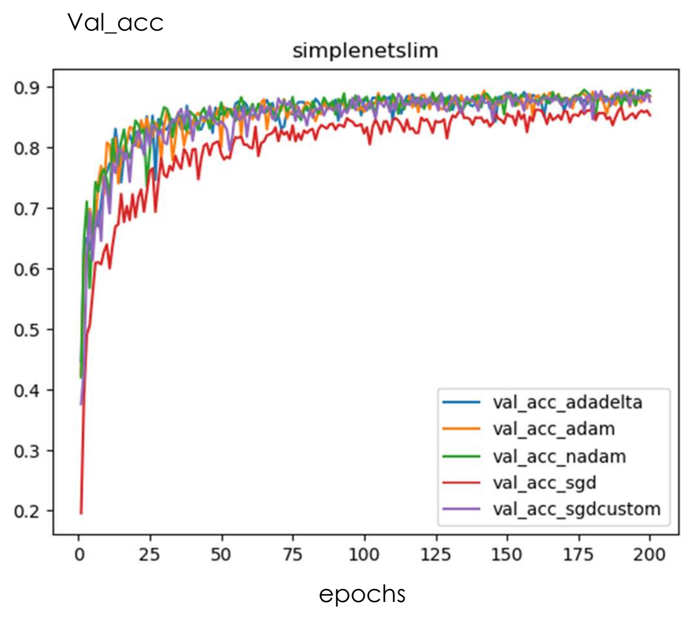
4.2. Optimization

Until now, we've covered the 3 base models. All 3 models weren't made specifically for embedded system use. Naturally, an optimization sounded possible. We went through 3 possibilities of changing the models to compare each model with and without these changes. Once the best characteristics for a model are found, comparisons are made between the 3 models to know which one has the best accuracy.

4.2.1. Optimizer

The optimizer can be changed in any model implemented on Keras in order to have the best possible training. The aim of the optimizer is to minimize the loss function and avoid the local minima. It is important to know that not all optimizers are adapted for each model. 4 famous optimizers and a modified one were tried on our models to compare if one is clearly better than others. The 5 optimizers are: Adadelta, Adam, Nadam, SGD and one called Sgd_custom implemented to have an adaptative decaying learning rate which was fit after experiments.





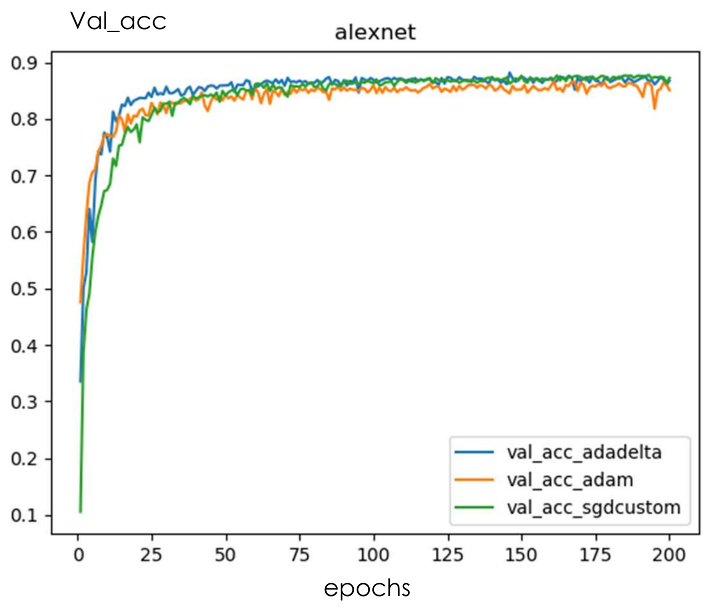


Figure 4 – Comparison of the different Optimizers

According to available papers [9], Adam is apparently the best for many models so it was the most important one to try. SGD was also chosen because of its wide use across papers covering convolutional neural networks.

For the AlexNet model, SGD and Nadam were not successful since the accuracy was not rising over 10%, therefore they were removed from the graph.

For SimpleNet and SimpleNetSlim, only one optimizer is way behind the others: SGD, which will not be used later in the project. For AlexNet, it is clear that Adadelta and SGD_custom are more efficient than Adam.

For a clearer comparison between models afterwards, only one optimizer must be chosen. Only 2 of these optimizers are chosen: Adadelta and SGD_custom.

According to papers where Adadelta doesn't suit a lot of models, SGD_custom was finally chosen to compare models on other characteristics.

4.2.2. Batch Normalisation

Batch normalisation is a technique for improving the performance and stability of neural networks. The idea is to normalise the inputs of each layer in such a way that they have a mean output of zero and standard deviation of one, similarly to how the inputs to networks are standardised. Since the network is a just series of layers where the input of one becomes the input of the next, instead of normalising only the input of the network, it is possible to normalise the input of layer.

The initial intention behind batch normalisation is to optimise the network training by normalising the activations of the previous layer for each batch. Batch Normalisation has been shown to have several benefits [10]. For example, it makes weights easier to initialise, makes more activation functions viable and allows higher learning rates. But the main benefit is the quicker training of the network. At first glance, we could think that batch normalisation slows down the training because of the extra normalisation calculations during the forward pass and the additional hyperparameters to train during the back propagation. However, the network should converge much more quickly so training should prove to be faster overall.

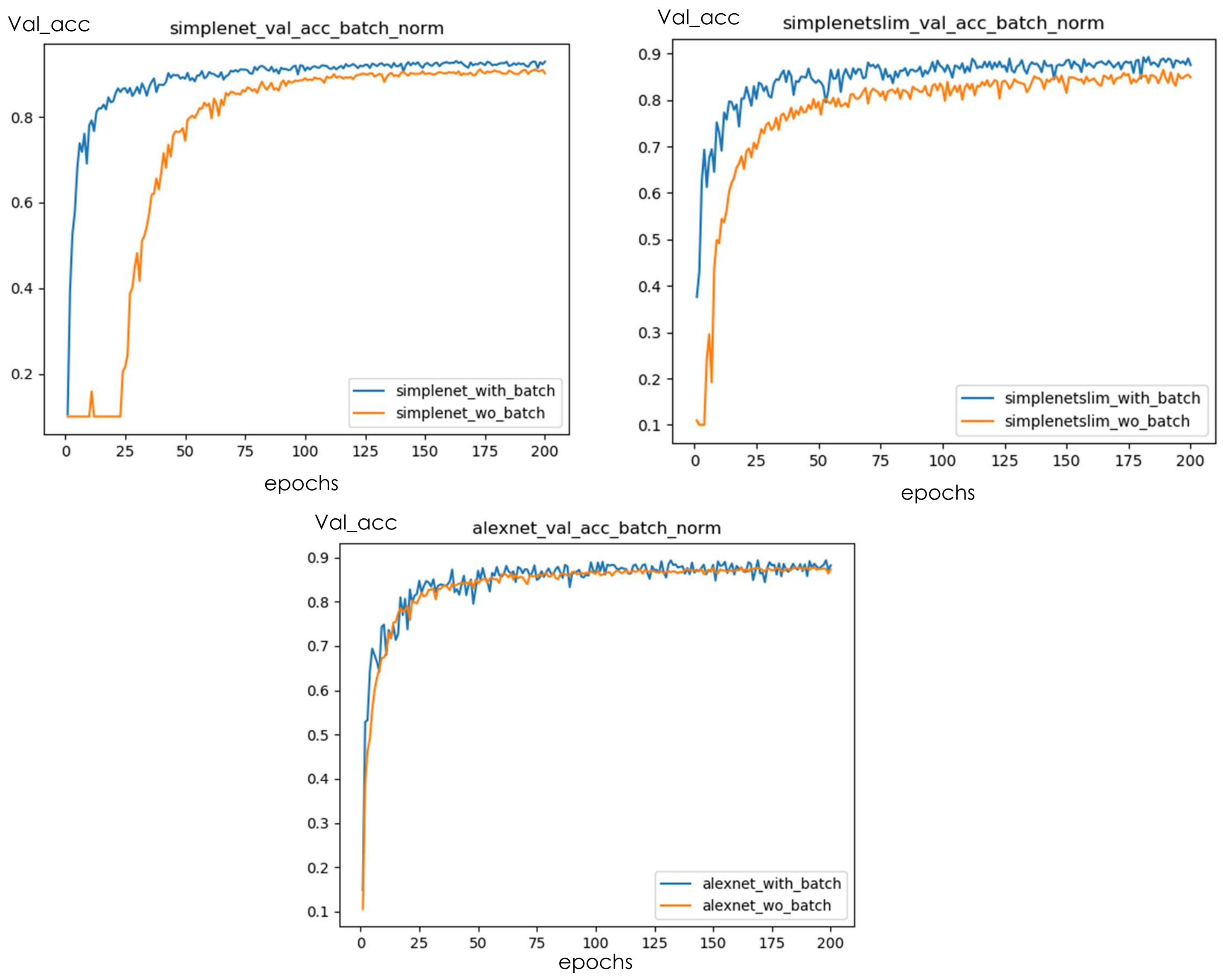


Figure 5 - The effect of Batch Normalisation

The global trend we can observe for all 3 models is how the peak validation accuracy is always higher when there's batch normalisation compared to when there isn't. The validation accuracy of both SimpleNet and SimpleNetSlim converge much quicker with batch normalisation as well.

These results explain the need for batch normalisation and why it is applied during all of our tests later on.

4.2.3. Dropout

When a dataset is small, like CIFAR-10, a model can quickly overfit. To avoid that, a Dropout layer can be added. This method randomly drops nodes to 0 (the portion of dropped nodes is defined by the argument of Dropout Function) during training. This simulates having a number of different networks. Many papers prove that Dropout is considered as a very effective way of reducing overfitting and improving the generalization of the model when testing on new images. However, its efficiency is apparently only visible on big networks.[11]

4.2.4. L2 Regularisation

L2 Regularisation is another technique of regularisation very similar to Dropout. The main difference remains in the fact that it does not drop random nodes to 0 but instead forces the weights to be smaller. The calculation is the sum of the squared values multiplied by the argument of the function, which is often set to low values. However, its efficiency is worse than Dropout for big Networks. For our case and according to [12], regularisation is not really adapted because of the big size of our networks but we still tried it in case SimpleNetSlim could benefit from it.

4.3. Model Compression

Model Compression is the main subject of our project since we want to use a model on an embedded device. This model will have to be the smallest, with the best possible accuracy. Different techniques of compression are available but in the allocated time, our work was focused on 2 popular techniques, Pruning and 8 bits Quantization while converting the model on TensorflowLite.

4.3.1. Pruning

Magnitude-based weight pruning is a popular compression technique based on eliminating non important values of a neural network [13]. Neural network parameters values are set to zero in order to remove low weights connections between the layers.

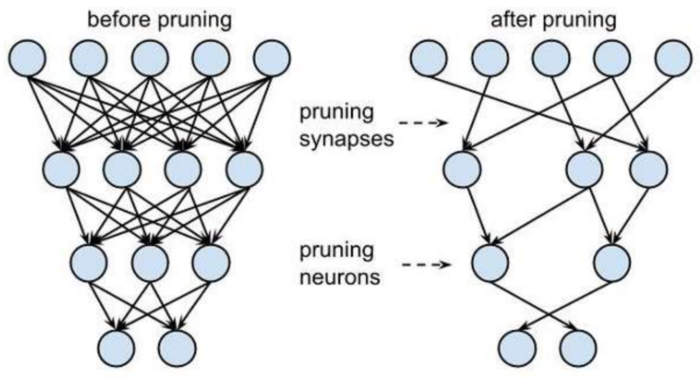


Figure 6 - Pruning

The Keras API removes, by itself, connections based on their magnitude and their importance in the training. The importance of pruning is determined by the portion of weights we want to keep. The less weights we have, the more the system accuracy falls. A compromise between pruning and accuracy has to be made. Once some weights have been deleted, the system is smaller and unnecessary computations are skipped.

The only constraint of this method is to, "only" and not other libraries, use TensorflowLite when putting the model on an embedded device, because it will take advantage of the pruning to make computations faster.

4.3.2. 8-bit Quantisation

When a model is saved in Keras, Weights are coded in 32-bit floats. A 32-bit float increases precision but increases the quantity of computations needed heavily. A Tensorflow Library allows the conversion of a model from Tensorflow to TensorflowLite and the reduction at the same time the size of coded weights. The best possible quantization is to go from 32-bit floats to 8-bit integers (values between -128 and 127) thus basically dividing the model size by 4 without losing too much accuracy.

5. Experimental Results

5.1. Setup

5.1.1. Dataset Used

There's a wide range of image recognition datasets available online we could have chosen. But since the goal is to conduct many small tests at first, a lightweight dataset like CIFAR-10 [14] making that easier only makes sense.

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) with 6000 images per class. There are 50 000 training images and 10 000 test images. The 10 classes are completely mutually exclusive. Meaning that there is no overlap between automobiles and trucks for example. Between the 50 000 training images contain exactly 5 000 images from each class.

It is possible to import the CIFAR-10 dataset with Keras using Python very easily since it's in the Keras library.

Data augmentation

To get a bit more of unusual images during training and adapt the system to changes and regarding the small number of images of the dataset, data augmentation is applied with 10% horizontal and vertical translations, and random horizontal flip.

Based on [15] and similar articles, data augmentation can significantly increase accuracy on training models, it has been used in that same way for every training in this project.

5.1.2. Toolbox

As stated earlier, all the experiments shown in this paper are run on a separate server equipped with an RTX 2080 Ti graphics card. The server runs the 1.14.0 version of Keras-Tensorflow with Python 3.6. Since it's an RTX, the server uses the CUDA toolkit to access the GPU in order to make use of the tensorflow-gpu 1.14.0 library.

5.2. Method results

5.2.1. Dropout

The Dropout has been added to SimpleNetSlim since it wasn't in the model to compare with and without. The argument is set to 0.2, that is to say 80% of weights are not dropped out.

On another hand, the dropout, which was at 0.4, has been deleted from AlexNet and SimpleNet.

As seen in the graphs of the figure 8, SimpleNet is a little more efficient with Dropout with a difference of 1% accuracy. AlexNet's curve is also more stable with Dropout. However, as predicted earlier, Dropout is not efficient on SimpleNetSlim since the model isn't that big. We can observe a loss of 3% accuracy when adding Dropout.

Some other experiments were done with changing the Dropout parameter but they were not conclusive in regards to the accuracy drop.

Dropout can only be effective on bigger neural networks such as AlexNet or SimpleNet but when the size shrinks, the efficiency decreases.

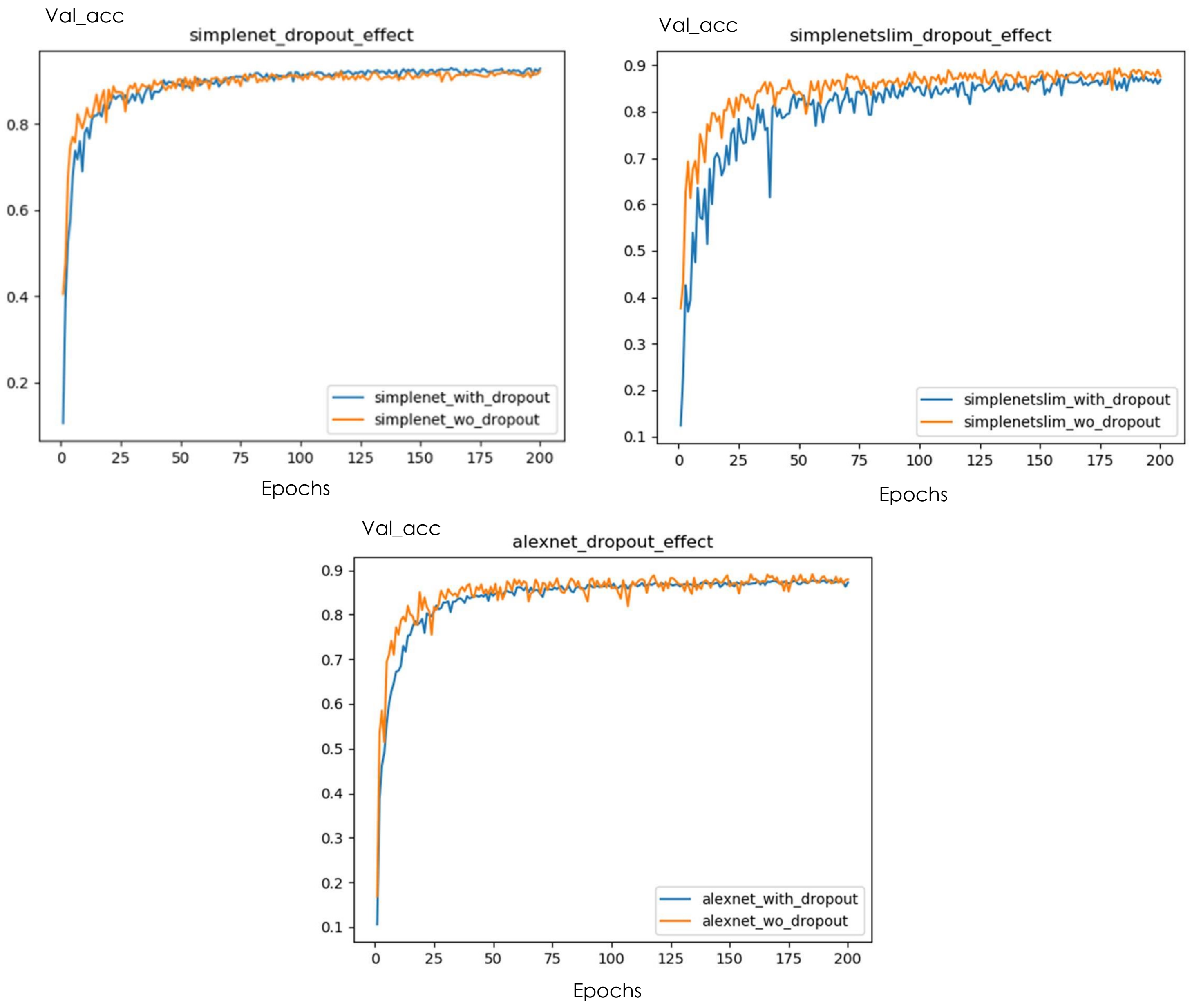


Figure 7 – Dropout Effect

5.2.2. L2 Regularisation

L2 Regularisation Layers were not implemented on basic models. We tried to add it on convolutional layers with an argument value of 0.001 to make very low values. However, as seen in the figure below, the results are poor with loss of accuracy in any case and very unstable validation accuracy curves in terms of accuracy. Therefore, regularisation was then not relevant to use later on the project.

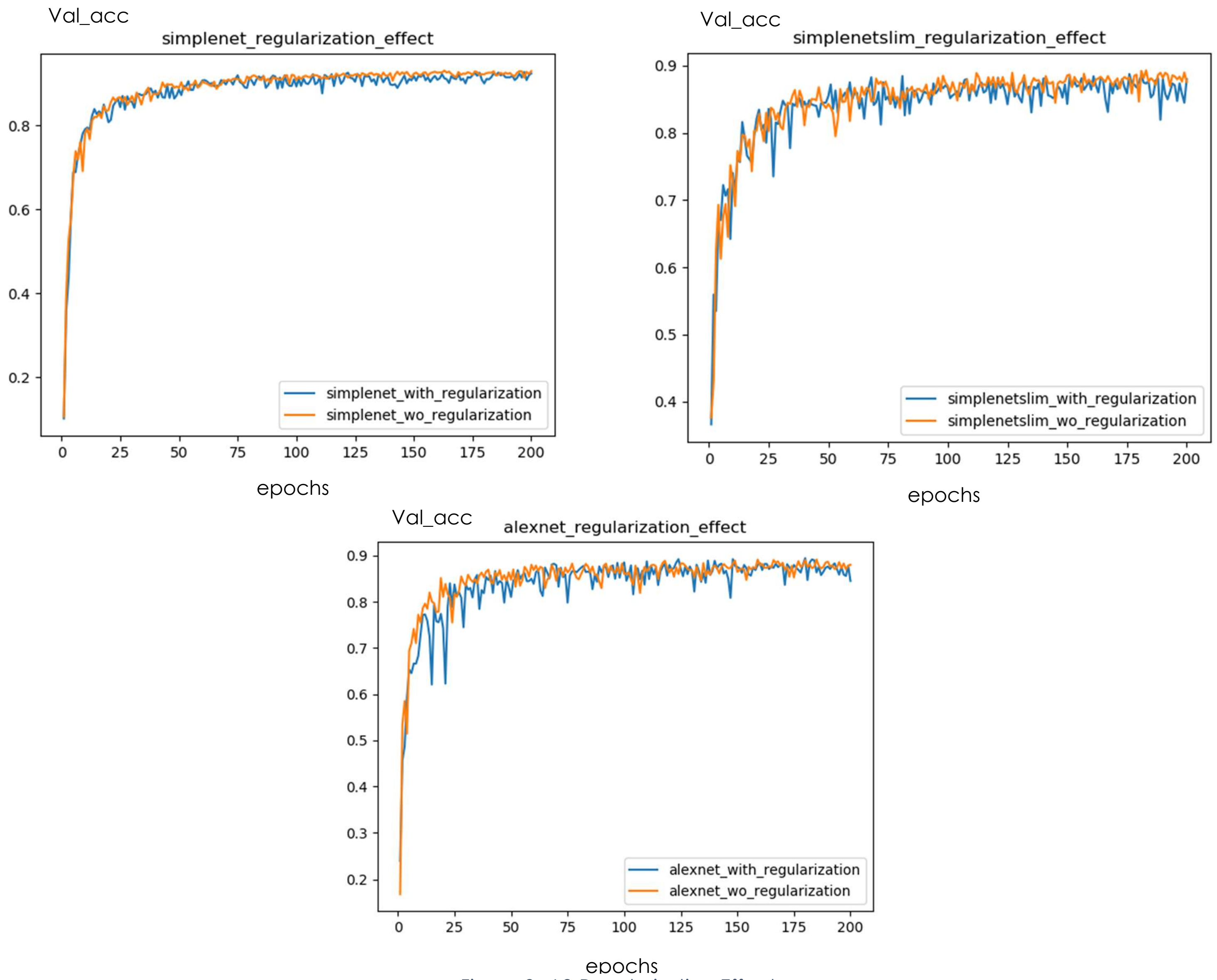


Figure 8 - L2 Regularisation Effect

5.2.3. Pruning

Pruning models have been created with a different sparsity each. Sparsity is defined as the portion of low magnitude weights deleted from the model. The first try was set at 95% and another one at 90%. During the try at 95%, the accuracy was not consistent since it decreased way too much with nearly 10% less than the original models even if the size was going down a lot.

The following table shows the results on size for 95% sparsity:

Model	AlexNet	SimpleNet	SimpleNetSlim
Unpruned Model size (Mb)	113.42	42.07	2.52
Pruned Model Size compressed (Mb)	7.68	2.91	0.18

Table 1 - Pruning effect on size with 95% sparsity

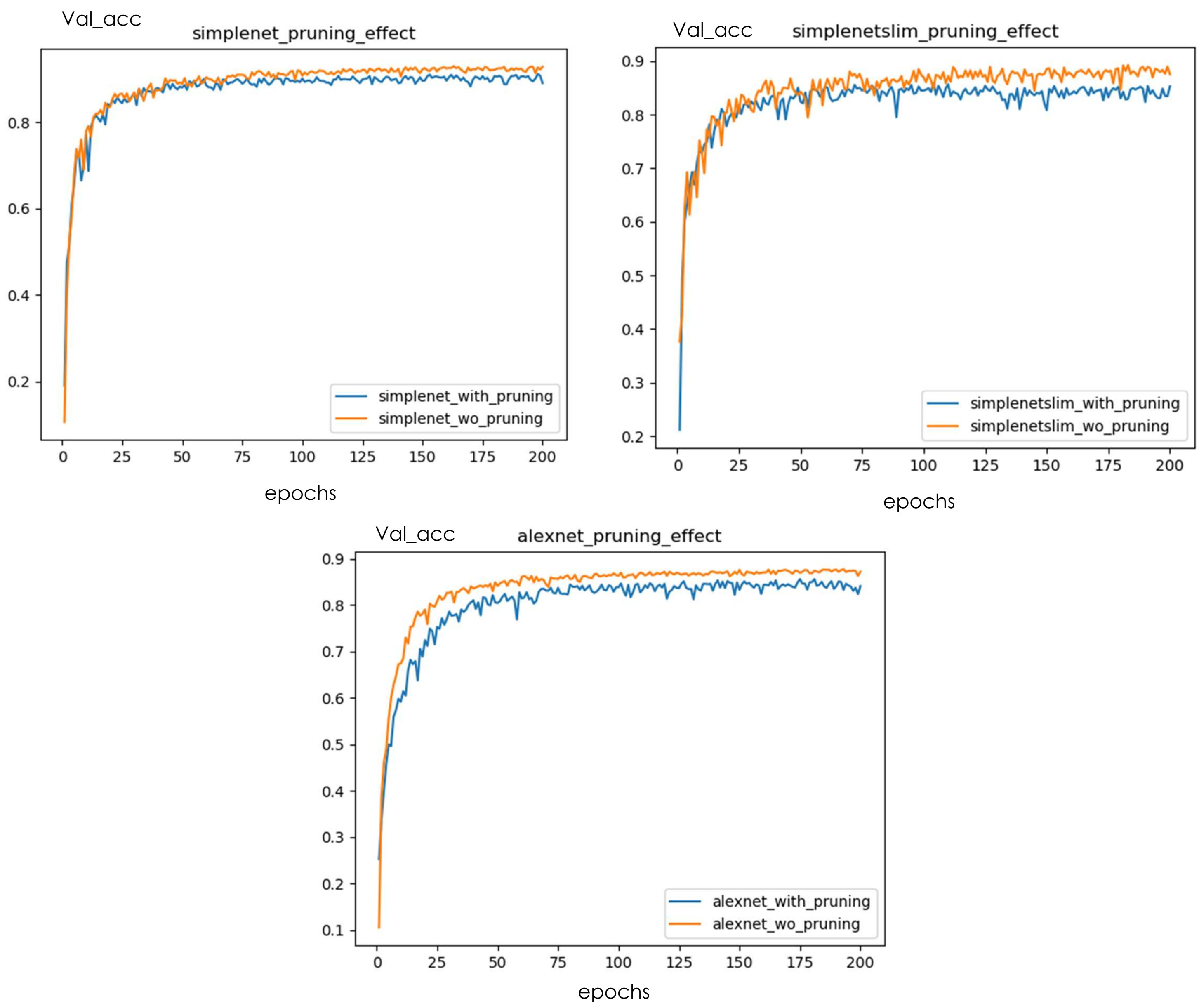


Figure 9 - Pruning Effect on validation accuracy with 90% sparsity

The second test with 90% sparsity produced the following results.

The pruning of the three models results in a significant loss of accuracy but has to be linked with the decrease in size of the models. For SimpleNet, the final accuracy dropped by 2%. For SimpleNetSlim and AlexNet, it dropped by about 2.5%. The following table shows the size reduction applied using pruning:

Model	AlexNet	SimpleNet	SimpleNetSlim
Unpruned Model size (Mb)	113.42	42.07	2.52
Pruned Model Size compressed (Mb)	10.92	4.10	0.25

Table 2 - Pruning Effect on size with 90% sparsity

5.2.4. 8-bit quantisation

As detailed in the Model Compression part, 8-bit quantization divides model's size by 4. The 3 models were successfully divided by 4 with this method but no tests were made on the new TensorflowLite model to know how accuracy dropped because of the lack of time.

5.3. Model Comparison

After studying, evaluating and optimizing each of the models, it is now time to compare their results. When evaluating the model, we can calculate 2 important comparable performance measures. The 2 mentioned measurements here are the accuracy and precision. Accuracy is the most intuitive performance measure and is simply a ratio of correctly predicted observations to the total observations. It is important to understand the expression "Correctly" predicted here. It means when the actual class is the same as the predicted class.

Precision, on the other hand, is the ratio of correctly predicted positive observations to the total predicted positive observations for one class. This evaluates how precise the model is by dividing the number of correctly predicted observations by the sum of the latter and the number of incorrectly predicted observations. (an incorrectly predicted observation is when the predicted class is positive while it actually is negative) [16]. Then to obtain the precision value for an epoch, the mean precision of the 10 classes is calculated.

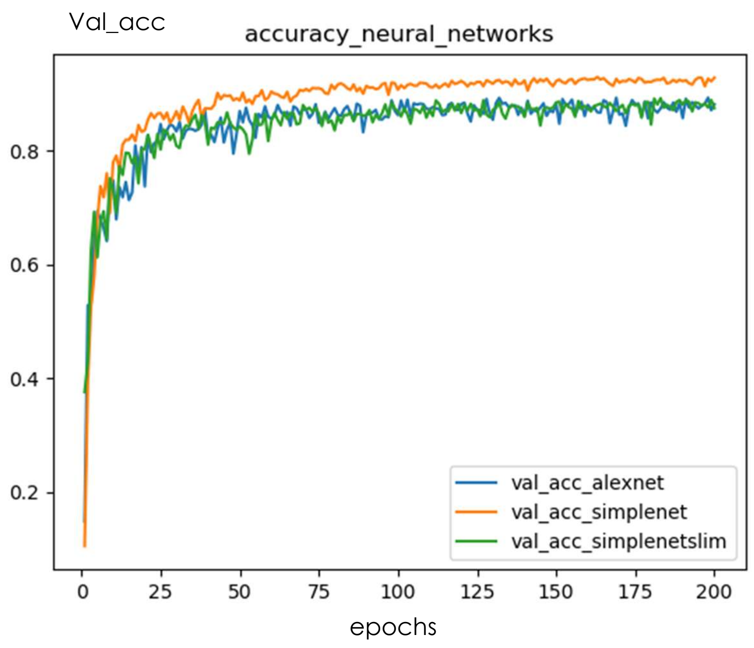


Figure 10 - Accuracy Comparison

The evolution of accuracy over 200 epochs is shown on figure 10. The final test accuracy value for each model is:

Model	AlexNet	SimpleNet	SimpleNetSlim
Accuracy	0.8744	0.9274	0.8828

Table 3 - Final Accuracy values

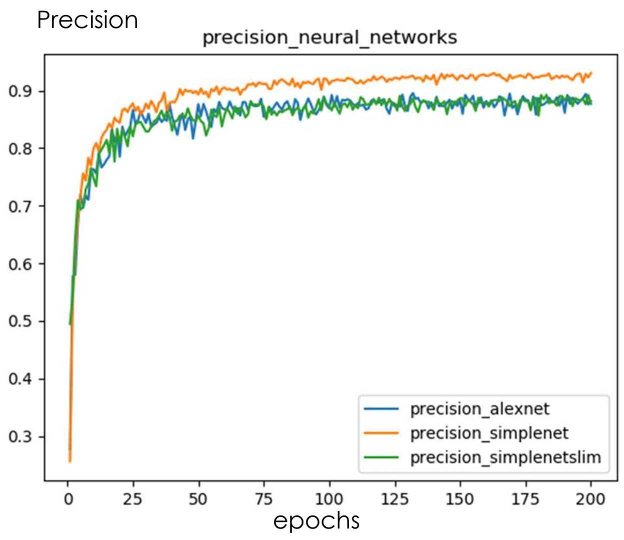


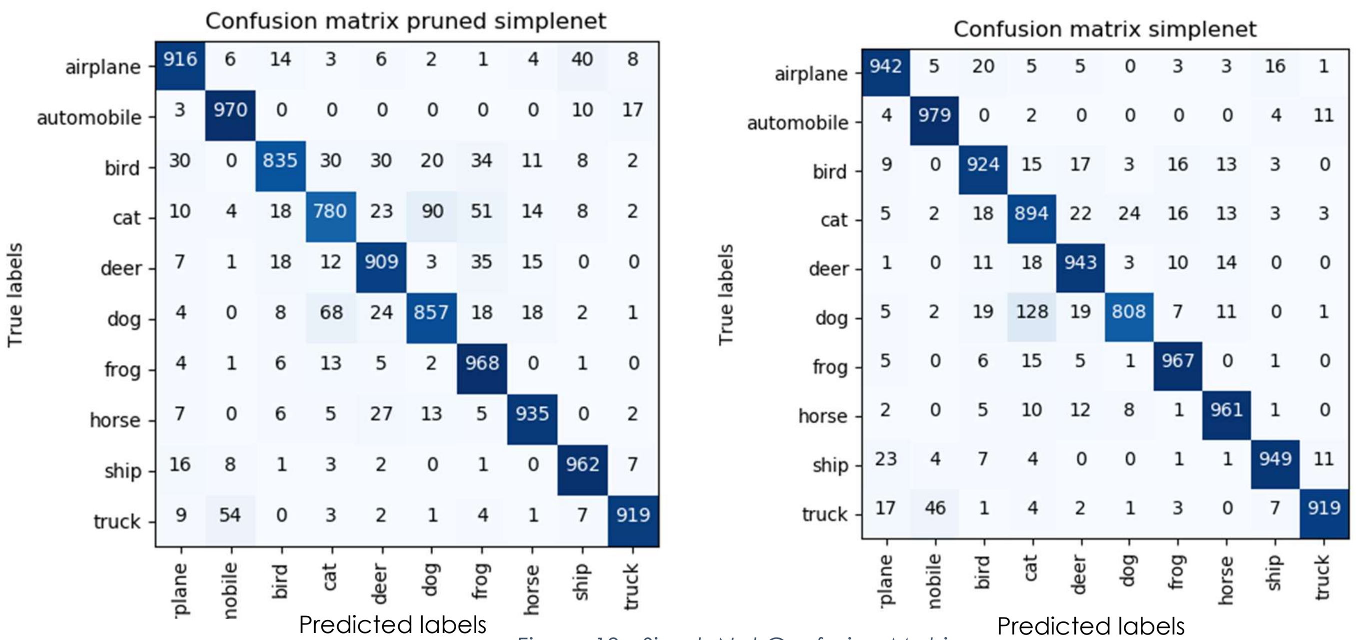
Figure 11 - Precision Comparison

The evolution of precision over 200 epochs is shown on figure 11. The final test accuracy value for each model is:

Model	AlexNet	SimpleNet	SimpleNetSlim
Precision	0.8733	0.9302	0.8768

Table 4 – Final Precision values

The confusion matrix for each model pruned and unpruned is shown next. The confusion matrix is a table layout representing the performance of an algorithm. Each column of the matrix represents the instances in a predicted class while each row represents the instances in the actual class. For example, the pruned SimpleNet confusion matrix shows 780 cats recognized out of 1000. The algorithm mistakes the cats for dogs the most as it recognized 90 dogs out of the 1000 cat images. In comparison, the SimpleNet confusion matrix shows 894 cats recognized out of 1000. While the numbers are never the same, this shows the effect of pruning on the performance of the same algorithm.



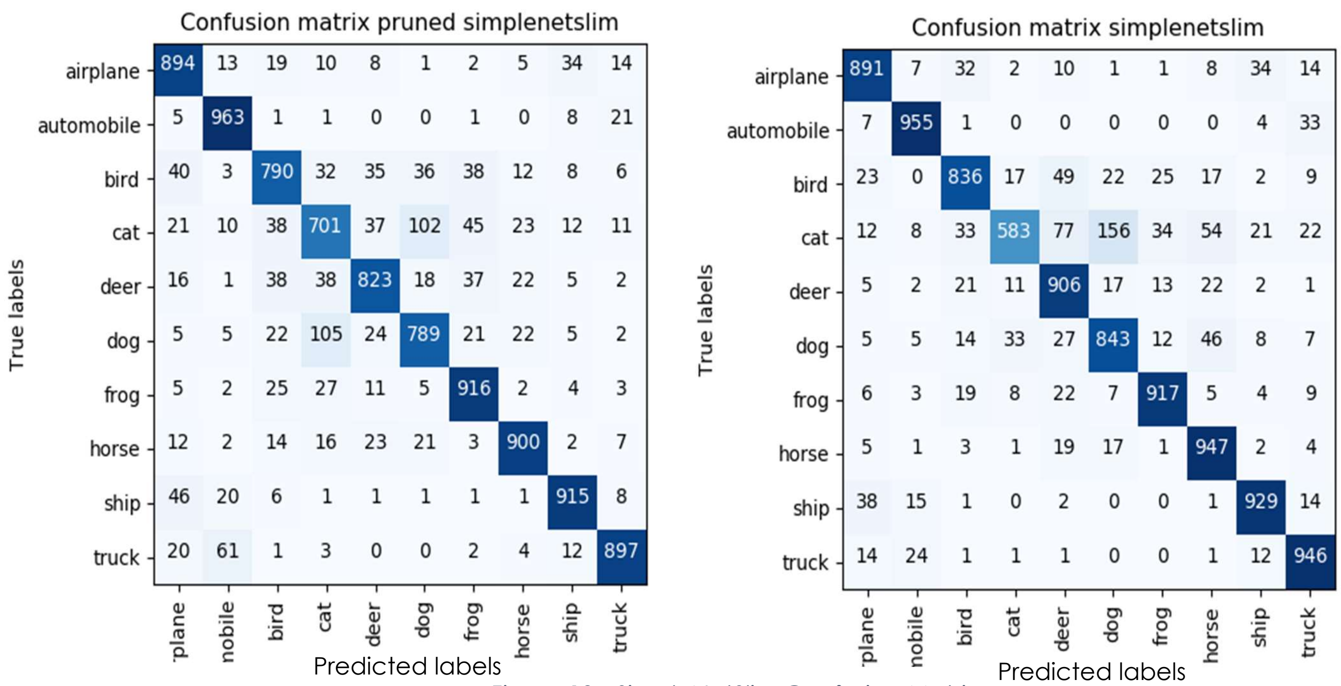
Confusion matrix pruned simplenet

True labels	'plane	nobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	916	6	14	3	6	2	1	4	40	8
automobile	3	970	0	0	0	0	0	0	10	17
bird	30	0	835	30	30	20	34	11	8	2
cat	10	4	18	780	23	90	51	14	8	2
deer	7	1	18	12	909	3	35	15	0	0
dog	4	0	8	68	24	857	18	18	2	1
frog	4	1	6	13	5	2	968	0	1	0
horse	7	0	6	5	27	13	5	935	0	2
ship	16	8	1	3	2	0	1	0	962	7
truck	9	54	0	3	2	1	4	1	7	919

Confusion matrix simplenet

True labels	'plane	nobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	942	5	20	5	5	0	3	3	16	1
automobile	4	979	0	2	0	0	0	0	4	11
bird	9	0	924	15	17	3	16	13	3	0
cat	5	2	18	894	22	24	16	13	3	3
deer	1	0	11	18	943	3	10	14	0	0
dog	5	2	19	128	19	808	7	11	0	1
frog	5	0	6	15	5	1	967	0	1	0
horse	2	0	5	10	12	8	1	961	1	0
ship	23	4	7	4	0	0	1	1	949	11
truck	17	46	1	4	2	1	3	0	7	919

Figure 12 - SimpleNet Confusion Matrix



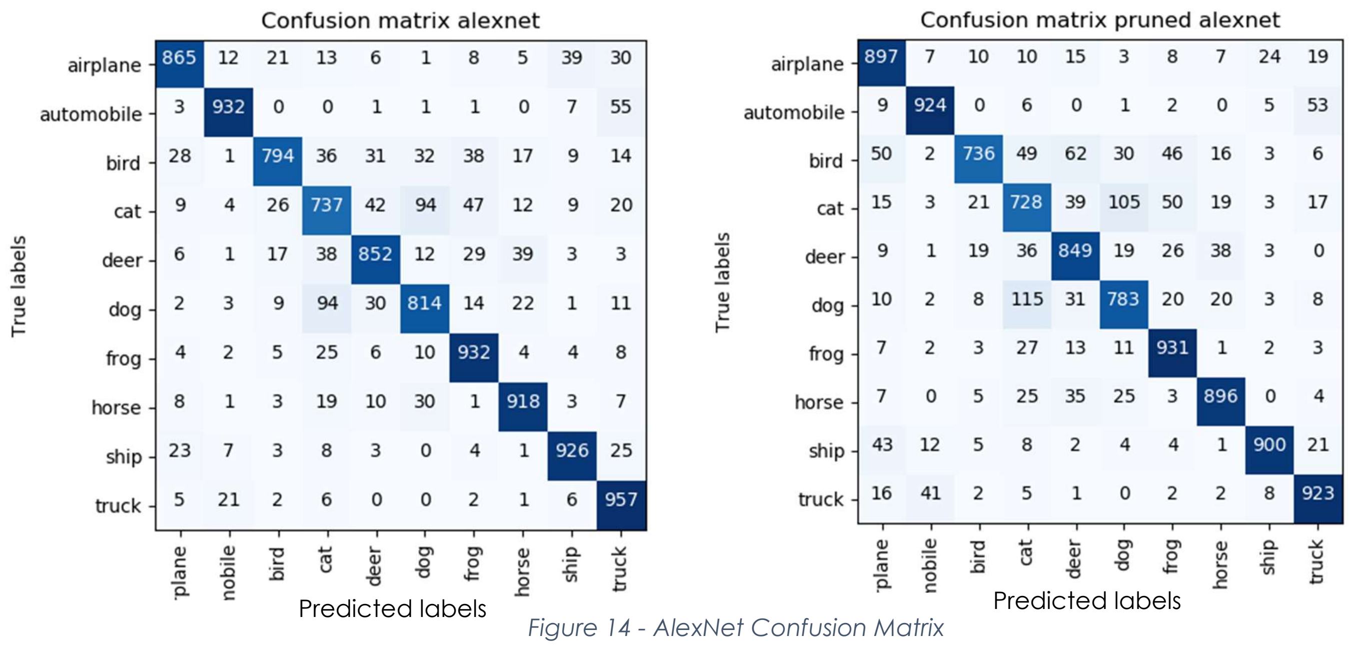
Confusion matrix pruned simplenetslim

True labels	'plane	nobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	894	13	19	10	8	1	2	5	34	14
automobile	5	963	1	1	0	0	1	0	8	21
bird	40	3	790	32	35	36	38	12	8	6
cat	21	10	38	701	37	102	45	23	12	11
deer	16	1	38	38	823	18	37	22	5	2
dog	5	5	22	105	24	789	21	22	5	2
frog	5	2	25	27	11	5	916	2	4	3
horse	12	2	14	16	23	21	3	900	2	7
ship	46	20	6	1	1	1	1	1	915	8
truck	20	61	1	3	0	0	2	4	12	897

Confusion matrix simplenetslim

True labels	'plane	nobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	891	7	32	2	10	1	1	8	34	14
automobile	7	955	1	0	0	0	0	0	4	33
bird	23	0	836	17	49	22	25	17	2	9
cat	12	8	33	583	77	156	34	54	21	22
deer	5	2	21	11	906	17	13	22	2	1
dog	5	5	14	33	27	843	12	46	8	7
frog	6	3	19	8	22	7	917	5	4	9
horse	5	1	3	1	19	17	1	947	2	4
ship	38	15	1	0	2	0	0	1	929	14
truck	14	24	1	1	1	0	0	1	12	946

Figure 13 - SimpleNetSlim Confusion Matrix


Confusion matrix alexnet

True labels \ Predicted labels	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	865	12	21	13	6	1	8	5	39	30
automobile	3	932	0	0	1	1	1	0	7	55
bird	28	1	794	36	31	32	38	17	9	14
cat	9	4	26	737	42	94	47	12	9	20
deer	6	1	17	38	852	12	29	39	3	3
dog	2	3	9	94	30	814	14	22	1	11
frog	4	2	5	25	6	10	932	4	4	8
horse	8	1	3	19	10	30	1	918	3	7
ship	23	7	3	8	3	0	4	1	926	25
truck	5	21	2	6	0	0	2	1	6	957

Confusion matrix pruned alexnet

True labels \ Predicted labels	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	897	7	10	10	15	3	8	7	24	19
automobile	9	924	0	6	0	1	2	0	5	53
bird	50	2	736	49	62	30	46	16	3	6
cat	15	3	21	728	39	105	50	19	3	17
deer	9	1	19	36	849	19	26	38	3	0
dog	10	2	8	115	31	783	20	20	3	8
frog	7	2	3	27	13	11	931	1	2	3
horse	7	0	5	25	35	25	3	896	0	4
ship	43	12	5	8	2	4	4	1	900	21
truck	16	41	2	5	1	0	2	2	8	923

Figure 14 - AlexNet Confusion Matrix

The previous results show that, globally, the performance isn't hindered too much. It is important to have really high accuracy but since the goal is compression and the difference isn't too impactful, it is safe to choose the smallest model in size, SimpleNetSlim, and to apply pruning on it.

5.4. Weights and Flops

In this project, the main subject is to run neural networks on embedded devices. It's really important to handle the number of flops and weights since they can affect the computation needed for training or testing any model. While doing so, they affect the energy consumption on the devices as well.

Weights are automatically calculated after the training of every model, it can be verified with calculations. For AlexNet, SimpleNet and SimpleNetSlim, parameters are 14,859,082, 5,497,226 and 312,682, respectively.

According to the architecture, FLOPS (Floating Point Operations per second) can be calculated too. For AlexNet, there are 1.80 e8 FLOPS. SimpleNetSlim is made of 5.58 e7 FLOPS and SimpleNet has 6.52 e8 FLOPS.

6. Conclusion

Multiple experiments and comparisons were done on three models. AlexNet was a benchmark as an example of a big and old models and how their architecture can be improved. The main objective of the project is to run a neural network on an embedded device using an application to be developed later on.

As it is described, even when pruned, AlexNet's size remains too significant to be put on edge devices. Moreover, its accuracy is too low compared to SimpleNet which has a smaller size. Even when comparing with SimplenetSlim, its accuracy is quite similar, but its size is 50 times bigger.

If we compare SimpleNet and SimpleNetSlim, the latter seems more appropriate to be used because of its size, computation power needed and accuracy. Even if the accuracy is a bit lower, the FLOPS used and size are really low and can be even lowered with Pruning and 8-Bit Quantization. A trade-off between some more accuracy and speed/size. The model we want to put on an embedded device has to be chosen with a particular focus on 3 characteristics: accuracy, size and computation power needed to run it. A compromise as to be done in order to reach the optimal network.

7. Future Work

Four subjects can be explored to complete a bit more the project.

- First, a comparison between 8-bit quantised models and non-quantised models has to be done. From recent works available online, the accuracy shouldn't drop too much because of the size of the 3 models studied but tests must be done to ensure that.
- Second, an estimate energy consumption according to weights and flops calculated for each model needs to be made.
- Third, the models need to be tested on different embedded devices (for example phones) to compare their efficiency on devices less powerful than the computer we used to train the models. Moreover, Energy consumption can also be verified directly on these devices.
- Finally, an application using one of those models can be developed to make more realistic tests of the power needed to run the application on a selected number of phones. The phones must be chosen according to their processing unit. In order to select a list of phones needed, a prior study of the available devices has to be made.

8. References

[1] A Survey of the Recent Architectures of Deep Convolutional Neural Networks https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf

[2] Let's Keep it Simple, Using simple architectures to outperform deeper and more complex architectures https://arxiv.org/pdf/1608.06037.pdf

[3] A Survey of Model Compression and Acceleration for Deep Neural Networks https://arxiv.org/pdf/1710.09282.pdf

[4] DeepSZ: A Novel Framework to Compress Deep Neural Networks by Using Error-Bounded Lossy Compression https://arxiv.org/pdf/1901.09124.pdf

[5] AI Benchmark: Running Deep Neural Networks on Android Smartphones https://arxiv.org/pdf/1810.01109.pdf

[6] http://ai-benchmark.com/

[7] ImageNet Classification with Deep Convolutional Neural Networks http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf

[8]https://github.com/Coderx7/SimpleNet

[9]How to pick the best learning rate for your machine learning project https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2

[10]Glossary of deep learning, Batch Normalization https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82

[11] Dropout: A Simple Way to Prevent Neural Networks from Overfitting http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

[12] An Analysis of the Regularization between L2 and Dropout in Single Hidden Layer Neural Network https://uksim.info/isms2016/CD/data/0665a174.pdf

[13]Magnitude based weight pruning with keras https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras

[14]https://www.cs.toronto.edu/~kriz/cifar.html

[15] Further advantages of data augmentation on convolutional neural networks https://arxiv.org/pdf/1906.11052.pdf

[16] https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/