OBJECT TRACKING WITH COMPRESSED NETWORKS

ALEXANDER ALFVÉN

Master's thesis
2016:E53

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

This thesis investigates the effects of network compression on machine learning networks developed for object tracking. As neural networks require considerable storage and memory, what is acceptable on larger computer systems may not be suitable for embedded implementations. More traditional algorithms for object tracking have non-trivial storage and memory requirements, and the question is if a deep regression based network for object tracking can become more viable after compression.

By using transfer learning on the networks Inception and Caffenet, which are originally trained for object recognition, it is found that a regression network for object tracking could be exceptionally robust to quantization. While one network fails to properly recognize the task, the other network successfully retrains on the task - and after the use of quantization demonstrates a near zero loss of accuracy. The resulting quantized network is just as effective at object tracking as the non-quantized network, but with reduced computational cost and just a quarter of the required memory.

Acknowledgements

I would like to thank Microsoft for the opportunity to do this Master's Thesis, and for supplying the hardware to carry it out.

I would also like to thank my Microsoft supervisors Johan Windmark and Gustav Träff for their guidance and support, along with Anders Stålring for his additional thoughts and input.

Finally I would like to thank my Lund Unversity supervisors Håkan Ardo and Mikael Nilsson for their support and guidance throughout the many steps of this thesis.

Contents

1 Introduction **7**
1.1 Previous Works . . . 7
1.2 VOT Challenge . . . 7
1.3 TensorFlow . . . 8

2 Background **8**
2.1 Datasets and Overfitting . . . 8
2.2 Dataset bias . . . 10
2.3 Used Dataset . . . 11
2.4 Dataset expansion . . . 11
2.5 Neural Networks . . . 12
2.6 Fully Connected Layers . . . 12
2.7 Convolutional Layers . . . 13
2.8 Pooling . . . 14
2.9 Regularisation . . . 14
2.9.1 Dropout . . . 14
2.10 ReLU . . . 15
2.11 Transfer Learning . . . 15
2.12 CaffeNet . . . 15
2.13 Inception . . . 16
2.14 Optimising Algorithms . . . 17
2.14.1 Gradient Descent . . . 17
2.14.2 Stochastic Gradient Descent (SGD) . . . 17
2.14.3 Adaptive Gradient Descent (ADAGRAD) . . . 18
2.15 Reduction and Quantization . . . 18

3 Implementation **19**
3.1 Dataset Processing . . . 20
3.2 Network Structure . . . 20
3.3 Converting CaffeNet . . . 22

4 Results **22**
4.1 CaffeNet . . . 24
4.2 Inception . . . 28

5 Conclusion **32**
5.1 Further Work . . . 32

1 Introduction

The aim of this master thesis is to investigate the effects of compression of a Neural Network designed for object tracking in a video. The purpose is to establish how this network compression could affect the viability in using neural networks on embedded (e.g. Mobile) platforms. Typically, neural networks are intended for deployment on PC platforms, where memory and compute capabilities are significantly greater than that on embedded platforms.

The size of most networks makes deployment on embedded platforms a challenge. Available resources such as memory is typically a fraction of PC platforms have available, and the cost of storage is often much higher as well. The GPUs used in embedded platforms are typically smaller and as a consequence more limited in their ability to handle complex computations. The size of the network also increases memory bandwidth requirements, which is also more limited. The use of Neural Networks for object tracking is not novel, although it is still a relatively recent one. However, previous implementations have not touched on network compression. It is therefore network compression and its effects on object tracking networks that this thesis will investigate.

1.1 Previous Works

Several neural network implementations demonstrated leading accuracy on the 2015 Visual Object Tracking challenge [KML+15]. However, those particular neural networks were not at all capable of real-time performance even on PC hardware. Fortunately, there are several other implementations that do not perform with quite the same degree of accuracy, but are significantly faster. In particular, this thesis will be built on the work set by the paper "Learning to track at 100FPS" [HTS16], which while not a leader in accuracy, demonstrated significantly faster performance.

The effects of compression on the performance and accuracy has been studied in other domains; Björgvinsdottir and Seibold in their paper "Face Recognition Based on Embedded Systems" study the effects in detail of network compression on embedded hardware in the face recognition domain [SH16]. While some of their findings should produce similar results in this thesis, such as performance, their results in accuracy are not guaranteed to fully translate.

1.2 VOT Challenge

The Visual Object Tracking Challenge is a yearly competition in which visual tracking systems can be compared in a repeatable and precisely defined manner. The primary goal is to compare and evaluate progress in the visual tracking field [KML+15].

The challenges themselves are graded on Accuracy, Robustness, and Speed. All of these metrics are naturally important; an implementation that is slow or sloppy may not be usable in reality. As such, even networks that do not lead in accuracy and robustness may still be of interest if they can maintain that level of accuracy with better speed than other implementations.

The VOT challenge as such provides a set of metrics to compare implementations; and can help researchers decide which areas of research could produce beneficial results.

1.3 TensorFlow

TensorFlow is an open source library by Google for use on Linux and Mac OS in Machine Learning [Inc16]. Neural Networks are not normally implemented fully by hand, this simply takes too much time. Instead, software libraries are used which have already implemented the operations involved in neural networks. This only leaves the programmer to define the shape of the network, making the process of creating and changing networks significantly easier and faster, and reduces the lines of code that would be required from thousands to a few hundred at most.

Furthermore, libraries such as these often support and handle GPU computation, and execution parallelisation across both local and remote hardware. TensorFlow itself is implemented in a mix of C++ and Python. The primary API is in Python, with only a limited API for C++ [Inc16]. GPU acceleration is implemented for NVIDIA's CUDA.

2 Background

Machine learning is the process of teaching a specific algorithm how to perform a desired task. Mitchell provides a concise definitions as follows: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[Mit97]

There are a vast number of possible tasks that such a program could be given. The perhaps most common one with images would be classifiers: where images contain a specific type of object (Pencil, Train, Car, etc), and the network is expected to tell you which of these are present in an image. The performance would then be measured by how often it predicts correctly. Experience would come from subjecting the network to images containing these types of objects, with the intent that it will learn to differentiate characteristics of these objects.

In this thesis, the task is instead to learn to quantify the translation. Since this is a task where a numerical output is expected, it is instead called a regression task [GBC16]. To solve a regression task for object tracking, where the new location is defined by the four edges of a bounding box, the learning algorithm is tasked to output a function:

$$f : R^n \implies R^4 \tag{1}$$

Performance is now instead defined by how close the network prediction is to the correct translation.

2.1 Datasets and Overfitting

When it comes to measuring performance of a network, we require the use of more than just a single set of data. Networks are subject to the experience of its training dataset for thousands if not millions of times: but this is not the data that the network will see once it is subjected to real world usage. The experience with the specific input the network has seen while learning will normally make it very good at handling that input; but the accuracy of the network for this data will not represent the accuracy of the network on other real use data [GBC16].

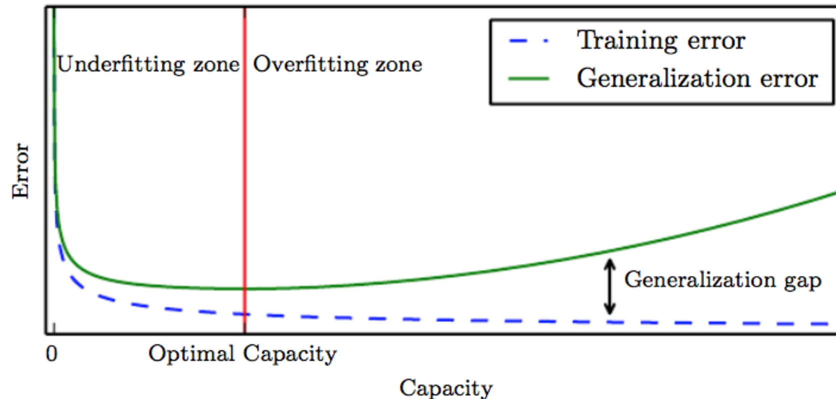

Figure 1: Network error as it changes with more training [GBC16]

Therefore in order to get an idea of how well the network performs on “real” data, it needs to be separated into several sets. The first, and by far largest, will be the training set. As the name implies, this is the data that will be used for teaching the network how to perform its task. Because of the need of much data as possible to properly learn - this dataset will typically make up 90% or more of the amount of data available. The remaining 10% will be used by two datasets used for measuring the estimated performance in real use scenarios.

The first of these datasets is the validation set, and this dataset is run periodically during training. The performance of the network on this dataset gives is an indication if the network is correctly learning to solve the problem, and it works as a guideline for adjusting the training process. Unlike the training set however, the network is not allowed to learn from the errors of the output. Every time the validation set is run, it will be as if the network has never seen these images before - which is the condition we need to expect in real world use.

The final set is the Test set, and this is only run once the network has completed training. Because adjustments may be made to the network as it trains, a final dataset is needed to verify how the network performs on data where we cannot have made adjustments for at all. This set will help tell if the changes made during training have exhibited certain biases.

As the network is trained, the accuracy for the training should continuously (if gradually) improve. Hopefully, the accuracy of the validation set will also improve; however this will not occur at the same rate as the training set. But as long as the performance of both the training set and validation set is improving, the network is learning and is on the right track.

While the network is able to benefit from additional training iterations, the network is said to be underfit. What it means is that the network has not yet been trained to generalise the input to its peak. Knowing that the network is underfit is important, so that we do not stop training prematurely. But this begs the question of when we should stop training.

One answer to that is when the network is no longer converging, when the accuracy of the validation set stops improving, often called early-stopping. It should be noted that the training set will likely continue to improve on accuracy.

However, this is not desirable - as this means that the network is no longer learning the general characteristics of the task, and is instead training only on specific and possibly irrelevant traits in its training data. While this could seem harmless, it is important to realise that this comes at the cost of performance to the validation set - that is, real world use.

Once a network has stopped learning how to handle the generalised problem, it has begun growing overfit instead. One can visually observe when a network transitions from underfit to overfit on a graph by plotting the accuracy of the network on the training and validation set, versus the time/iteration. The point where the curve for the performance of the validation set diverges away from the curve of the performance for the training set, the network has transitioned from being underfit to being overfit [GBC16]. This process is illustrated in Figure 1.

Typically, the network state is periodically saved, and continued use of the network should utilize the network in the state where it began to overfit. That a network has transitioned to being overfit does not mean that further training is impossible. But it does mean that the training process needs to be adjusted: either by adding in new batches of data, changing some of the network parameters such as learning rate, or adding additional regularisation.

2.2 Dataset bias

Neural networks learn by generalising the input problem, extracting common features that are present in its training set to find the commonality between the input that it sees, and the output that it expected. For example, when presented with a dog, it may learn to identify the presence of a tail, head, four legs, and ears. If the network is trained to recognise specific breeds of dogs, or even just to separate it from other animals that also have those parts such as cats, it can also learn these characteristics in more detail, so that it knows what a dog's head looks like and a cat's tail.

However, carelessly feeding a network with data may result in a network that does not quite train in a manner that is expected - or wanted. The network does not inherently know that it should learn to identify a cat; it has to learn this by being subjected to images where there is a cat, and images where there is not. Is it therefore important to make sure that these images are separated only by the presence of the cat. Not on the level of each input, but rather over the entire set.

That is, it is important to make sure that images that contain cats, do not also contain other key features. For example, if images that contain cats are all taken in the grass, and images without cats are taken indoors, then the network is unlikely to realise that it is the presence of the cat that it should learn from; and will instead focus on the presence of grass.

An example, though evidence whether it actually occurred or not is lacking, is a tank identification example. This particular neural network was intended to learn to detect the presence of a tank; but what the developers didn't realise was that the images of tanks they had been supplied were taken on a cloudy day [Inc16]. As a result; when the network was tested, it was not capable of identifying tanks within the image, but instead identified the weather.

The only real means of avoiding dataset bias is to make sure that the input data is sufficiently varied. Images of cats should be taken indoors and outdoors, in sunny and cloudy weather. The cats should be different cats, of different

Figure 2: The ILSVRC2015 dataset contains many videos of animals, such as this one where the goal is to track a squirrel

species and ages, and in different stages of activity. And these types of differences needs to be true when the network then comes to dogs, and horses, and all other objects it should be capable of identifying [Inc16] [GBC16]. In a tracking setup, a number of different objects are required, each travelling in variable directions and at variable speeds.

2.3 Used Dataset

For this thesis the "ImageNet Object detection from video" dataset from 2015, or ILSVRC2015 for short, has been used. The ILSVRC2015 dataset is designed for classification [RDS+15]; and contains a total of 30 different object classes, mostly various types of animals, but also a few vehicles. However, the videos also come with object location data which can be used for training object trackers. Each video contains only one tracked object. Video quality is variable, with video resolutions ranging from 480x360 to 1280x720. Not every frame is manually annotated; only every 5th frame has manually entered data, with the remaining frames instead using interpolation.

2.4 Dataset expansion

Availability of training data for machine learning applications has seen considerable improvements in recent years. This in combination with improved hardware, are the primary reasons use of machine learning for various applications has increased significantly. But regardless of the size of the dataset, there are still significant benefits from increasing this dataset in artificial ways. With imagery, the means of doing this would involve making multiple variants of the source images where each image has been subjected to a number of modifications. These modifications can include the image being mirrored, rotating the image, applying noise, or even slightly changing the colour curves of the image [GBC16].

The purpose of these modifications is that the network is then further forced to learn to recognise the the problem it is being tasked to solve, rather than

individual images. In addition, some of these modifications may further aid in teaching the network to deal with less ideal images; the sort of lower quality images that regular users, lacking good photography experience, may be taking. Unlike classification case however, this thesis has an additional ways of expanding the dataset: by simply doing more comparisons than just the sequential frames in the video dataset.

Instead of simply comparing only Frame 5 versus Frame 6, we can also compare Frame 5 to Frame 7, 5 to 8, and so on. Not only does this provide more data, but it also gives the network opportunities to train in larger and more extreme translations that would typically be present when only looking frame-to-frame.

2.5 Neural Networks

In Machine Learning, "Neural Networks" comes from the similar term in neuroscience. Neural Pathways are a series of neutrons whose activation form a linear pathway. Neurons only activate when subjected to specific stimuli, and through a neural network can be created that when subjected to specific imagery, lights up a chain of neurons that form a specific neural pathways [SH16].

In machine learning, groups of neurons form a collected layer. How these neurons are connected to other neuron varies, and is commonly defined by the type of layer the neuron is located in [SH16] [GBC16].

A neural network does not possess any knowledge when it is created. Just like a real brain, a neural network needs to be trained to be able to carry out the task intended. This training is carried out by subjecting the network to large amount of different stimuli and gradually adjusting the parameter of each neuron to achieve the desired response. Given enough time and data, the network will eventually learn to generalise the problem, and produce the desired response from any appropriate input, not just for the input it has been subjected to during training [GBC16]. As a result, a classifier can learn to identify that there is a panda in any image that contains one - just like a human can recognise when seeing a panda.

In practice of course, neither humans nor artificial neural networks will be flawless. As both will typically have learned how to recognise a vast amount of objects, both can also be confused or mislead by what they are seeing - generating the wrong response. For artificial neural networks, the best counter to this is simply more data, data that is as numerous and diverse as possible. In practice of course, there is only a finite amount that exists, which will always place some constraints on just how good the networks can become.

2.6 Fully Connected Layers

In a fully connected layer, each neuron in the layer has a connection to all of the neurons of the input. A fully connected layer can be realised as a Matrix Multiplication between the input vector V and the weight matrix W of the local neuron, followed up by vector addition with the neuron's bias B. Fully connected layers are followed with an activation function F, which will be described later [GBC16]. The output is therefore generated by the equation:

$$(VW) + B \tag{2}$$

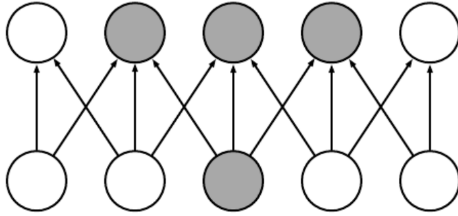

Figure 3: Convolutional neurons are only connected to a fraction of the input [GBC16]

The size of the output of this layer is configurable. While the breadth of the weight W does need to be of the same length of the input V, the length of the weight is configurable. The length of the W matrix defines the number of vectors it contains, and each of these vectors is called a feature. By adding additional features, each feature can become trained to activate on a different type of input, just like that of a neuron. The obvious drawback with increasing the number of features in the layer is that it increases the storage requirement and computational requirement of the layer accordingly. As such, while a layer will require multiple feature vectors, it is undesirable for the network to contain more than what is needed to satisfactory solve the problem. Because each vector should respond differently to inputs, the weight vector cannot be initialised with uniform number. If that was the case, then all of the vectors/features would always produce identical responses. A common workaround for this is randomly initialising the weights [GBC16].

2.7 Convolutional Layers

Convolutional layers are the second key layer type. Unlike fully connected layers where every input interracts with every output, convolutional layers only have each output interact with a small amount of the input, as shown in figure 3. Convolutional layers are useful for detecting localised features. Since neurons in convolutional layers only look within a small region, they will only activate on the presence of the desired feature in that region. Convolutional layers exhibit translation invariance, and can classify patterns regardless of the location within the input [GBC16].

A key realization of convolutional layers is that the weights used by the kernel of all neurons is shared. Unlike fully connected layers, convolutional layers can have a very small kernel size; and as these kernels are shared throughout the layer, the amount of parameters to store can be reduced by several orders of magnitude [Mur12].

With a 2-dimensional input image of I and a Kernel K, with output coordinates represented by (i, j) a convolutional layer can be implemented in the form [GBC16]:

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(m, n) K(i + m, j + n) \quad (3)$$

Because the neurons summarise regions, there will by default be overlap between adjacent regions. This is seldom necessary information. Convolutional

layers therefore have a stride which indicates the distance between the centre of each kernel. A stride other than one has considerable performance and memory benefits. Aside from reducing the amount of computations needed for the current layer, strides other than one also reduces the size of the output. With a stride of two, the size of the output is reduced by half.

The output of the layer will then be a form of feature map where a strong output indicates the presence of a specific feature within the region a neuron corresponds to. In conclusion, convolutional networks are useful for identifying important elements in the image.

2.8 Pooling

Convolutional layers are frequently followed by a pooling layer. These layers simplify the output generated by the convolutional layer, by summarising local output, which is what is then passed on to the next layer. Pooling has the consequence of making the network less sensitive to very minor translations in its input. Since pooling layers reduce and summarise regions, they also result in a smaller network [GBC16].

Typically used pooling functions are Max-Pooling and Average pooling. As their names imply, a max-pool function outputs the maximum value from within an region, wheras an average-pool function outputs the average value from a region. For example, for a kernel containing the values $[1, 3, 2, 1]$ the max-pool function will only output the value 3. For a region size of 2-by-2, a complete input of the size 20-by-20 is reduced to the size of 10-by-10.

It may seem counterproductive to use pooling when translation information is important. However, bear in mind that the translation information lost is only very fine. This minor loss of information should be compensated by the improvements in the networks ability to process the general features of the input images, and the considerable reduction in computation and memory requirements. Only a slight loss in attaining pixel accuracy is expected.

2.9 Regularisation

Regularisation are any modifications made to a learning algorithm where the goal is to reduce the generalisation error, that is the ability to solve the general task. Reducing the training error is not a goal of regularisation [GBC16]. There are many different types of regularisation techniques, some that can be very effective for specific types of tasks. For this thesis however, only some of the more common general-purpose regularisation techniques will be used.

2.9.1 Dropout

Dropout is a regularisation process of removing outputs from a preceding layer, in order to force the remainder of the network to utilise output that were not removed. This method is computationally inexpensive, and while it does slow down training slightly, it is still reasonably effective in improving the extent that the network can be trained.

The output from the input are removed at random, meaning that the network must learn to train in a manner where it cannot assume that specific input values are present. In effect, it forces the network to develop redundancy, as it

prohibits the network from training in a manner where it is heavily reliant on a small group of neurons. This in turn should improve the attainable accuracy of the network. In most implementations, this simply involves setting the output of the removed neuron to zero [GBC16].

The exact probability of keeping a neuron is itself a tunable hyperparameter, and different networks or even layers can use different dropout rates.

2.10 ReLU

Rectifier function, a simple but effective activation function that only outputs positive values. The function is defined as

$$f(x) = max(0, x) \tag{4}$$

ReLu has demonstrated good results in multiple networks in recent years. [Mur12]

2.11 Transfer Learning

Training a network from scratch is a process that for many tasks cannot be done quickly. While a simple network may be possible to train in a couple of hours, networks of those types often only solve very simple tasks (and may have more efficient alternative methods). Training more complex networks, such as those used for various image recognition tasks, can take days or weeks. Since more complex tasks require a network of greater size, training that network will take significantly longer. Yet many such complex tasks are not entirely unique. A network trained for face recognition for example, will have its earlier layers dedicated to identifying generalised shapes and appearances of a human face. Training a new network to do this would take significant amount of time, but thankfully, this is not needed [SH16] [Inc16].

Through the use of transfer learning, it is possible to take a network that has been trained in one problem and apply it to another problem. Obviously, some similarities between the two are needed, but it is for example possible to train a face recognition network that has learned to identify a group of faces, to instead identify a different group of faces. But even slightly more different applications are possible; one can also take a network trained for image classification, and retrain it to recognise object translation[HTS16].

This is done by taking an existing network that has been trained on a particular problem, and extracting its earlier layers. Specifically, the layers of interest are the convolutional layers - leaving out the fully connected layers. The result that these layers produce are then fed into a new set of fully connected layers, and it is these and only these layers that will be trained. In this thesis, two pre-trained networks are used: Inception and CaffeNet.

2.12 CaffeNet

CaffeNet is a model designed for ImageNet classification [Don]. CaffeNet itself is largely based on the network AlexNet [KSH12], with only a minor rearrangement of a few operations [Don]. As the name may suggest, the CaffeNet model is designed for use in with the Caffe library, and as such some conversion

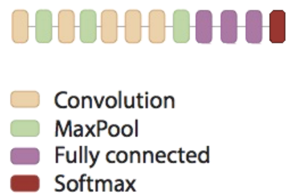

Figure 4: The CaffeNet network

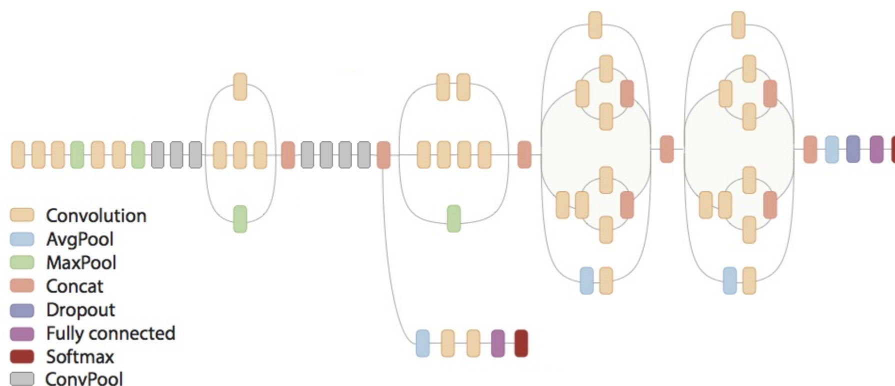

Figure 5: Layout of the Inception V3 network [SVI+15]

work is required to use it with Tensorflow. This conversion process is described in the implementation.

CaffeNet has a fairly simple model. As can be observed in figure 4, it has only a small amount of layers.

CaffeNet takes an input of $227 \times 227 \times 3$. The output prior to the first fully connected layer produces an output vector with a of length 9216.

2.13 Inception

Like CaffeNet, Inception is a model designed to process and classify images in the ImageNet dataset, but is built for use in Tensorflow. Inception designed to heavily use Convolutional Layers, making use of only a single Fully-Connected layer at the end [SVI+15] unlike the three fully connected layers used in CaffeNet. This has the benefit of making Inception require much fewer parameters and results in a network size that is smaller than CaffeNet.

Inception uses an input of $299 \times 299 \times 3$, and the output prior to the fully connected layer is a vector of length 2048.

For the sake of simplicity in reading figure 5, a block ConvPool represented by figure 6 is used to represent a grouping of layers that occurs multiple times within the network.

For the full specifications on each individual layer such as their input and output dimensions, see the Inception paper [SVI+15].

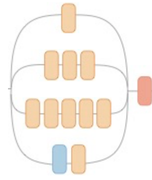

Figure 6: Inception ConvPool block

2.14 Optimising Algorithms

In machine learning the purpose of the learning algorithm, like in statistical estimation, is to for a given loss input L optimise its parameter θ in order to then minimise the sum of L across all iterations i. L is defined by the input X, the expected output Y, and all the network parameters θ [GBC16]. The resulting function as such has the form:

$$L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^i, y^i, \theta) \tag{5}$$

There are multiple approaches to this problem, so only the basic Stochastic Gradient approaches and some enhanced variants used in this thesis will be described.

2.14.1 Gradient Descent

Gradient descent is an iterative optimization algorithm for finding the local minimum of an input function. To do this, for every iteration the current point θ is changed by the negative gradient ∇ of the function L at its current point [Bö]. However, because these changes can be very dramatic, the magnitude of the proposed change needs to be limited. This magnitude is defined by a positive scalar value called the learning rate ϵ [GBC16][Mur12]. Using gradient descent from the current point θ a new point θ' is therefore obtained using the equation

$$\theta' = \theta - \epsilon \nabla_\theta L(\theta) \tag{6}$$

Tuning the learning rate requires care, as if the learning rate is too low it will take a very long time to train the network. However if the learning rate is too high, the network can fail to converge.[Mur12]

2.14.2 Stochastic Gradient Descent (SGD)

Stochastic gradient descent is an extension of the Gradient Descent method [GBC16]. The general idea behind SGD is the same as for gradient descent. However the SGD approach realizes that the calculated gradient is only an expectation that can be approximated using only a subset of the inputs. As for a given input size of M the computational cost of the gradient descent algorithm is $O(M)$, standard gradient descent becomes prohibitively expensive as more input is added. For the SGD, only a fraction of M is used; typically set as a constant for a computational cost of $O(1)$. A training set with billions of examples can therefore consist of updates that are generated by only a few hundred input samples [GBC16].

2.14.3 Adaptive Gradient Descent (ADAGRAD)

Standard SGD requires a considerable amount of effort in order to find suitable values for learning rates. ADAGRAD is a method of incorporating knowledge of observed data in earlier iterations of a training network in order to dynamically adjust the learning rate of the network. This has the benefit of reducing the amount of time and effort needed to find suitable hyperparameters to get the network to converge. Unlike SGD, it is much less sensitive to the initial learning rate being set too large or too small, as it will re-adapt the learning rate to better suit that of the network. [DHS11]

2.15 Reduction and Quantization

As the complexity of a network grows the storage requirement of it increases considerably. While there is a considerable storage cost to a network, the bigger issue is the memory costs of storing the network when it is being used. Most networks will when trained take up hundreds of megabytes in RAM and disk space. While typical PCs are able to handle this, for embedded devices that may not even have a gigabyte of RAM, it is too much. In addition, the size of the network puts considerable demands on memory speed and bandwidth, typically significantly more limited at as well. As a consequence, even a device with good computational capabilities may be throttled due to the memory requirements.

To alleviate this, we can reduce the network by converting it to a less complicated (and less accurate) data type. Intuitively, converting a 32-bit float to an 8 bit int should yield a size reduction by 75%. A network that would take 200mb of main memory, would take up just 50mb - a far more reasonable amount, and one that embedded devices are much more often prepared to handle. By remembering the minimum and maximum of each feature, it is then possible to specify where within this range each individual value lies. This approach makes it possible to remain much closer to the original value than a naive conversion to a lower precision type [War16]. In addition, speed can also be improved by using these 8-bit values for computation. Higher precision floats take a considerably longer amount of time to process than lower-precision floats or integers. In addition to this, embedded devices often have very limited high precision processing capabilities, and are instead designed primarily for low precision computations. By converting the network to use lower precision values, the hardware can be better utilised.

There is one potential drawback to doing the processing and not just storage in 8-bit however, and that comes to systems that do not have dedicated hardware ALUs or techniques for computing these types. For these systems, the hardware will have to reconvert the operations back to using 32-bit floats while the network is running. For these systems there will not be an improvement in speed, and because of the performance cost to re-convert the network, some performance is actually lost versus the non-reduced network. For these systems, it would be better to retain the original operations and perform the parameter conversions only when the network is being loaded.

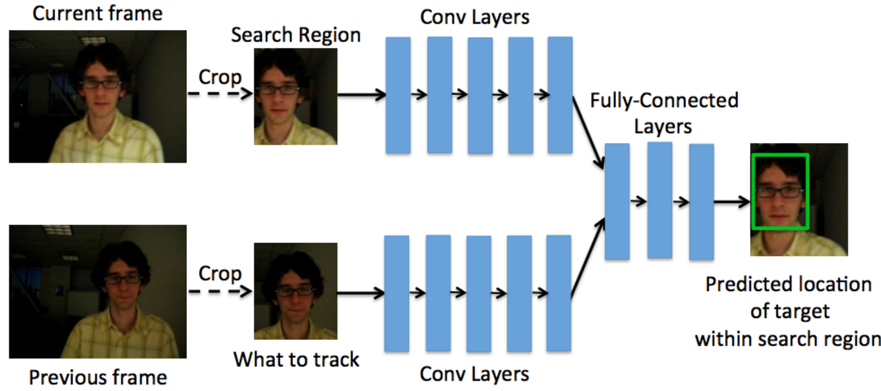

Figure 7: Full structure of the network, two images are processed separately by the convolutional stage and their outputs are then merged into a single vector to be processed by fully connected layers [HTS16]

3 Implementation

For tracking an object, this paper will base its method on the paper published by Held, Thrun, and Savarese [HTS16]. In order to be able to track an object, the object must first be designated. This is done by defining a bounding box that encompasses the object or region to be tracked.

For the input, two images are supplied. The first image is where the location of the object is known; the second is the following frame, where the location is unknown.

Because searching through fully sized frames would be immensely demanding, and largely unnecessary in a video, the network is not supplied with the full frame. Instead it is only given a cropped frame, centred on the bounding box of the object [HTS16]. The cropping region is defined by the distance from the middle of the bounding box to each edge, multiplied by 2. Note that for rectangular bounding boxes, this will produce a rectangular crop. The same cropping region applied to the starting frame is then applied onto the following frame. The purpose of the margin from the boundary box to the edge of the image is to provide a search region for the network to find the object within. A larger margin can handle a larger translation, but may result in some loss of accuracy and detail in the input.

The network structure as defined in the paper by Held, Thrun, and Savarese and is shown in Figure 7 [HTS16].

Finally, these resulting crops will need to be inserted into the network. Because the networks used in this thesis accept only specific input resolutions, the input has to be resized in order to fit. Since the input aspect ratio may not match the networks input aspect ratio, this rescale will not be uniform. The key point is that input bounding box will always be in the same regions, and that will define to the network the object being tracked.

3.1 Dataset Processing

Despite that the chosen dataset (ILSVRC2015) is intended for classification with location, some frames still do not have location data. Some videos have no location data at all. As a result, the entire dataset cannot be used. Fortunately, the dataset comes pre-divided into a training and validation set [RDS+15]. No further adjustments need to be made to this division, and they are kept as they are.

Because of the way the image will be cropped, there are additional steps that are taken when using the dataset. In order to ensure that there is some usable data adjacent to each of the bounding box edges, pairs where the bounding box either starts or ends close to the edge will be filtered out. How close the bounding box can be to the edge can be specified by a tolerance value T. Computing whether an image satisfies the tolerance involves, for both dimensions of each image, the shortest distance D from the bounding box edge to the image edge, divided it by the size S of the image for that dimension. This resulting value must exceed the tolerance value T in order to be used within the dataset. In conclusion, in order for a pair to be accepted into the dataset, it must for both dimensions of both images satisfy the equation

$$\frac{D}{S_{image}} > T \tag{7}$$

Its similarly important to ensure that the translation in the image pair does not place the resulting bounding box outside the second image. If that is the case, the pair is also rejected. In order to increase the amount of data available, the dataset is also expanded by not only looking at sequential frames. In order for the network to learn to potentially handle larger translations, image pairs will also compare each frame with subsequent 5 frames that contain manual annotations. This not only provides means to handle larger translations, but it provides the network with significantly more data. The result of this filtering and data expansion is that the datasets now look as follows:

	Pairs	Videos
Training Set	346566	1524
Validation Set	56367	221

Table 1: Number of image pairs and videos after filtering

3.2 Network Structure

The layout of the network in this thesis is divided into two main parts. The imported section, and the trainable section.

For the first section, the imported section, the network receives two input images I, which are separately run through the convolutional layers extracted from an imported network C. The output is then concatenated into a single

$$
\begin{aligned}
&[FullyConnectedLayer] && L_1 = (VW_1) + B_1 \\
&[Relu] && R_1 = F_{Relu}(L_1) \\
&[Dropout] && D_1 = F_{Dropout}(R_1) \\
\\
&[FullyConnectedLayer] && L_2 = (D_1 W_2) + B_2 \\
&[Relu] && R_2 = F_{Relu}(L_2) \\
&[Dropout] && D_2 = F_{Dropout}(R_2) \\
\\
&[FullyConnectedLayer] && L_3 = (D_2 W_3) + B_3
\end{aligned} \tag{9}
$$

Figure 8: The structure of the trainable part of the network

flattened vector V [HTS16].

$$
\begin{aligned}
V_{first} &= C(I_1) \\
V_{second} &= C(I_2) \\
V &= \{V_{first}, V_{second}\}
\end{aligned} \tag{8}
$$

These layers are not subjected to further training; and as such will always produce the same output. When training, a useful optimization is therefore possible. Instead of running each image pair through the full network every single time, the output from this section can be saved. Only needing to run through the convolutional layers once saves a significant amount of computation time, particularly when using a larger pre-trained network such as Inception.

Once the vectors V from the pre-trained network have been received, they are inserted into the second part of the network; the part of the network that will be trained.

For this, three fully connected layers are used, L1, L2, and L3. After the layers L1 and L2, a Relu function, and dropout is used. Each of the fully connected layers have their own Weight W and Bias B. W is a two-dimensional matrix with the length of the input for the layer, and breadth equal to the specified feature count for the layer. The Bias is a one dimensional matrix equal to the breadth of the layer's weight matrix.

As is the norm, the network bias is initialised to zero. The weights are initialised with a normalised random distribution. The standard deviation specified for this distribution is defined with N as the length of the input to the layer as follows:

$$
\sqrt{\frac{2}{N}}. \tag{10}
$$

The motivations for this choice of distribution are specified in a paper by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun [HZRS15]. Because this part of the network is generated and trained, there are now also hyperparameters to manage. The three main parameters that need to be managed is the number of features for each weight, the dropout rate, and the learning rate.

For this thesis, a feature count of 3000 was used for all three fully connected layers. For the dropout layers, a rate of 0.5 was used. Finally, for the Adagrad

optimiser, a learning rate of 0.0005 was used, and its initial accumulator value was set to 0.9. The batch size to use is tuned to the specific capabilities of the machine performing the training, though for this thesis a batch size of 400 is used. The pairs used in each batch is randomly selected.

3.3 Converting CaffeNet

CaffeNet is not originally released for the TensorFlow library. To use the CaffeNet model in TensorFlow the model needs to be converted. For this thesis, this was done using a CaffeNet-Tensorflow converter [Eth]. While the tool was effective in converting the network variables, the generated network definition ends up being more complicated than necessary. As a result, the network graph is manually defined, and the weights and biases generated by the converter are simply imported into their corresponding layer. The resulting network was saved to a file and verified in TensorBoard.

4 Results

As to be expected, the process of quantizing the networks has significantly reduced their storage requirement, as can be seen in table 2. Whereas previously the CaffeNet based network required nearly 400MB, it has been reduced to requiring just under 100MB. The Inception based network is smaller still.

For both networks, the storage requirement is similar to that of a smartphone or tablet application. For other embeded uses, the reduced size of these networks allows them to fit on much cheaper 128MB storage units, instead of requiring a costlier 512MB storage unit. As both networks have been reduced by the same ratio, after quantisation the size difference between the two networks has been reduced to 17.5MB instead of the original 73.2MB. This makes the original size advantage of Inception less benificial than previously, and how the networks perform in testing instead becomes more important.

	Normal	Quantized
CaffeNet	377.7 MB	94.5 MB
Inception	304.5 MB	77.0 MB

Table 2: Size of the networks before and after quantization

To measure RAM usage, the Python memory profiler is used. To avoid issues with garbage collection; the garbage collector is forced to run after initialisation of the network. The results listed in table 3 is the peak memory usage of the networks averaged across multiple runs. The overall memory usage when running is consistently within a few percentages of the peak, and can be interpreted as the effective memory requirement. For both networks, memory requirements are reduced by approximately 50%.

To measure how well the networks perform, they are subjected to four different benchmarks. One is a simple speed benchmark, while the remaining three evaluate the accuracy of the network.

The first benchmark measures the speed of the network, by testing the number of pairs the network is capable of processing per second. This is effectively

	Normal	Quantized
CaffeNet	1600 MiB	830 MiB
Inception	1450 MiB	750 MiB

Table 3: Peak memory during execution

the frame rate of the network. Because no suitable hardware was available for performing speed measurements however, and because the intent is to get an approxiate idea of how well the network should behave on embedded hardware, the speed benchmark is instead run on an Intel 4258U CPU.

The first accuracy benchmark measures mean accuracy over the course of an entire video. This allows gauging if the network has an easy or hard time with a particular video, and produces a result that is insensitive to the length of the dataset videos.

The second accuracy benchmark measures the mean accuracy on a per-frame basis by taking the mean error over the entire dataset. This produces a figure which measures how well the network performs overall.

The third accuracy benchmark is much like the second benchmark, except that instead of using the mean error of a frame, the bounding box edge with the greatest error is used instead. This can allow detecting deviations that could otherwise be masked when the frame mean is used.

The actual error of the three accuracy benchmarks is measured on the cropped and resized input. Since the scale of the input does not match the scale or aspect ratio of the source, the result does not measure the extent of the error as seen on the original frames. Additionally, since the Inception network has a higher input resolution, at least a slightly higher error would be expected.

Due to possible limitations with the quantization module used by Tensorflow, the speed of the CaffeNet network suffers severely when quantisized. Since the TensorFlow quantization module is quite recent, the means of fixing could not be made within the given timespan for the thesis.

As a result, benchmarking has not been possible to perform on the full validation and training sets due to the extreme amount of time this would require when benchmarking the quantized networks. The benchmarking sets have instead been reduced to run only on the first 200 valid pairs within the validation set, and the first 20 valid pairs within the training set.

	Pairs	Videos
Training Set	26011	1524
Validation Set	12179	221

Table 4: Number of image pairs and videos using during network evaluation

While this problem causes the speed difference to be non-representative, it does not affect the resulting network size or accuracy. While not possible to demonstrate the speedup on this particular network; the potential speed increases have already previously been demonstrated[SH16][Har].

Due to the limitations of time, the accuracy of the non-quantized networks has not been fully optimised, further improvements to the training techniques

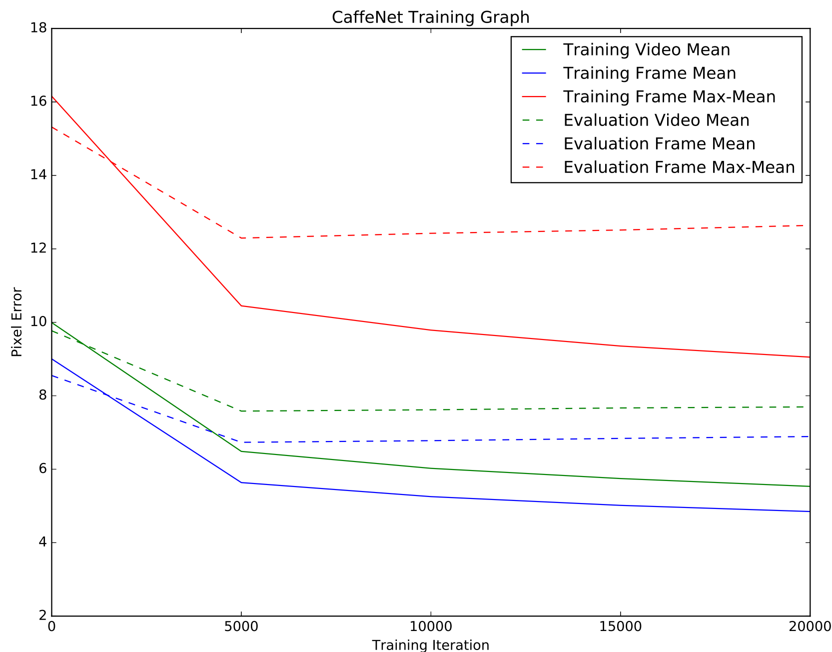

Figure 9: Accuracy of the CaffeNet model as it trains

and resulting network accuracy is still possible.

Note that because no changes were made to the networks as they trained, a test set is not used or required.

4.1 CaffeNet

The CaffeNet implementation is as expected a solid performer. The same implementation as demonstrated in the basis paper, the model appears to be reasonable at object tracking and requires little training time.

As can be observed in Figure 9, it takes only 5000 steps with a batch size of 400 for the network to reach optimum fitness. After that, the network ceases to improve on the validation set, only attaining gradually improving scores on the training set. As can be observed in the tables below, the performance of the network barely suffers at all. For both the training set and validation set, the average error is virtually the same.

Looking at the results in tables 5 and 6, it is clear that the network performs almost identically after compression. While it is strange that the validation set scores better than the training set, this is likely due to the smaller amount of samples used for the validation set. There are several further observations that can be made from the data.

	Normal	Quantized
Speed (Pairs/Sec)	5.88	0.52
Mean Video Error (Pixels)	8.91	8.91
Mean Frame Error (Pixels)	8.37	8.38
Mean of Max Frame Error (Pixels)	15.31	15.32

Table 5: CaffeNet training set results

	Normal	Quantized
Speed (Pairs/Sec)	5.88	0.52
Mean Video Error (Pixels)	7.76	7.77
Mean Frame Error (Pixels)	6.81	6.82
Mean of Max Frame Error (Pixels)	12.50	12.50

Table 6: CaffeNet validation set results

In particular, figures 10 and 11 indicate that very accurate frames do suffer slightly, with a notable decrease in the number of frames that have a average or maximum error of less than 1. Consequently, the number of videos with a mean error of less than 1 is also decreased.

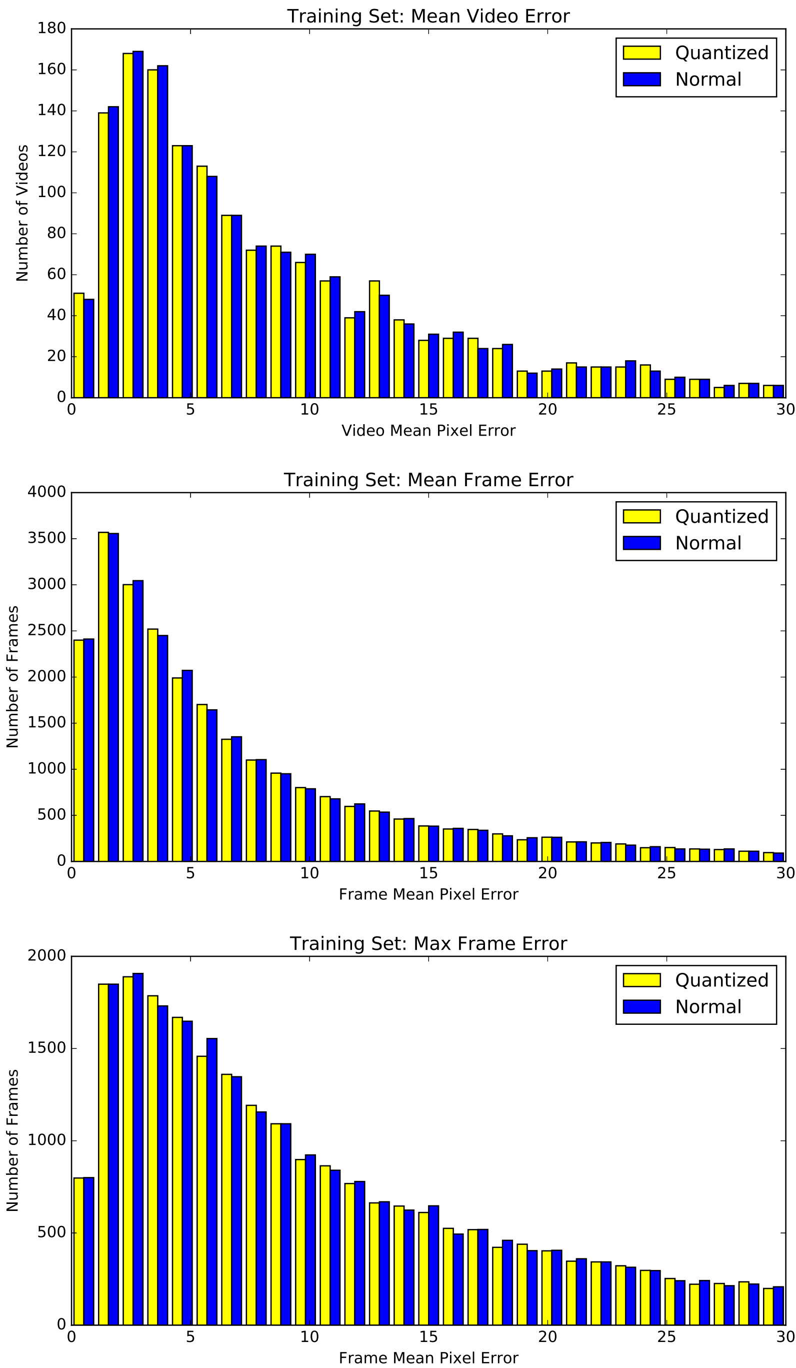

Figure 10: The CaffeNet model handles quantization well, with little difference to the accuracy on the training set

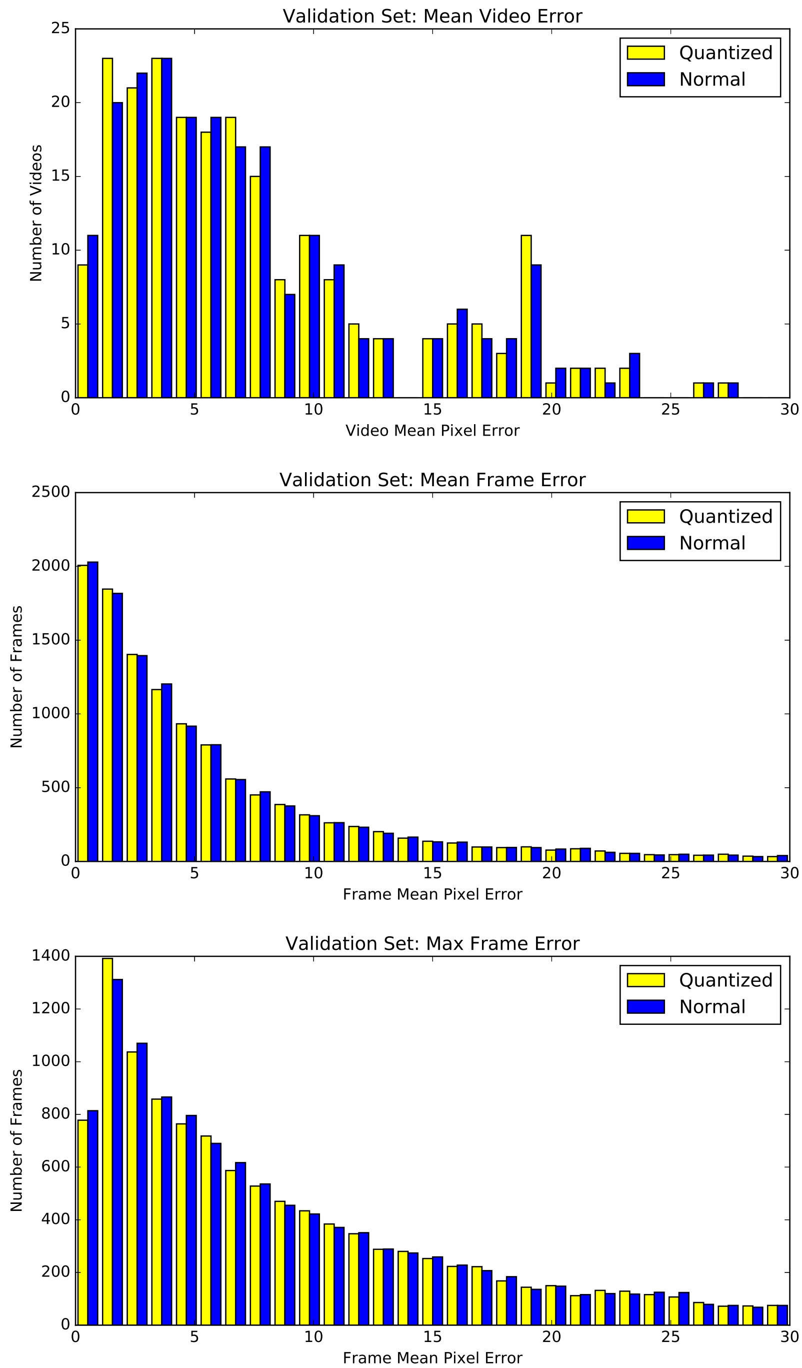

Figure 11: The CaffeNet validation set also shows minimal losses in accuracy after quantization

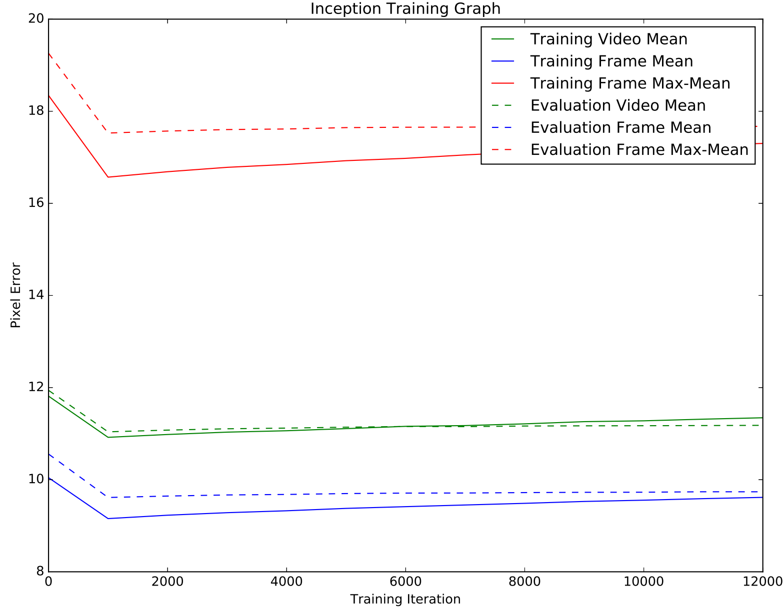

Figure 12: Accuracy of the Inception model as it trains

4.2 Inception

For object tracking, it would appear that Inception's approach of utilising a large amount of convolutional layers may be non-optimal.

As can be observed in tables 7 and 8, the Inception based implementation takes considerably longer to process each image. At just a single frame per second, even when considering that the network is CPU based, this is far too slow for use on embedded devices. To make matters worse, the accuracy of the network does not compensate for this decrease in speed. The inception network, even after normalising for the greater input size is not accurate, and struggles to properly track a given object, as can be observed in figures 13 and 14.

	Normal	Quantized
Speed (Pairs/Sec)	1.03	1.12
Mean Video Error (Pixels)	5.23	8.24
Mean Frame Error (Pixels)	5.03	7.95
Mean of Max Frame Error (Pixels)	9.24	14.33

Table 7: Inception training set results

Overall, while the Inception network does manage to attain a slightly smaller footprint, the network struggles with learning to properly solve the problem. As

	Normal	Quantized
Speed (Pairs/Sec)	1.03	1.12
Mean Video Error (Pixels)	12.49	13.90
Mean Frame Error (Pixels)	11.13	12.48
Mean of Max Frame Error (Pixels)	19.89	22.28

Table 8: Inception validation set results

a result, the implementation is not very viable for use in object tracking. While it required just 1000 steps to reach its optimum accuracy, which can be reached in mere minutes, this is simply an indicator that the network is struggling with handling the task.

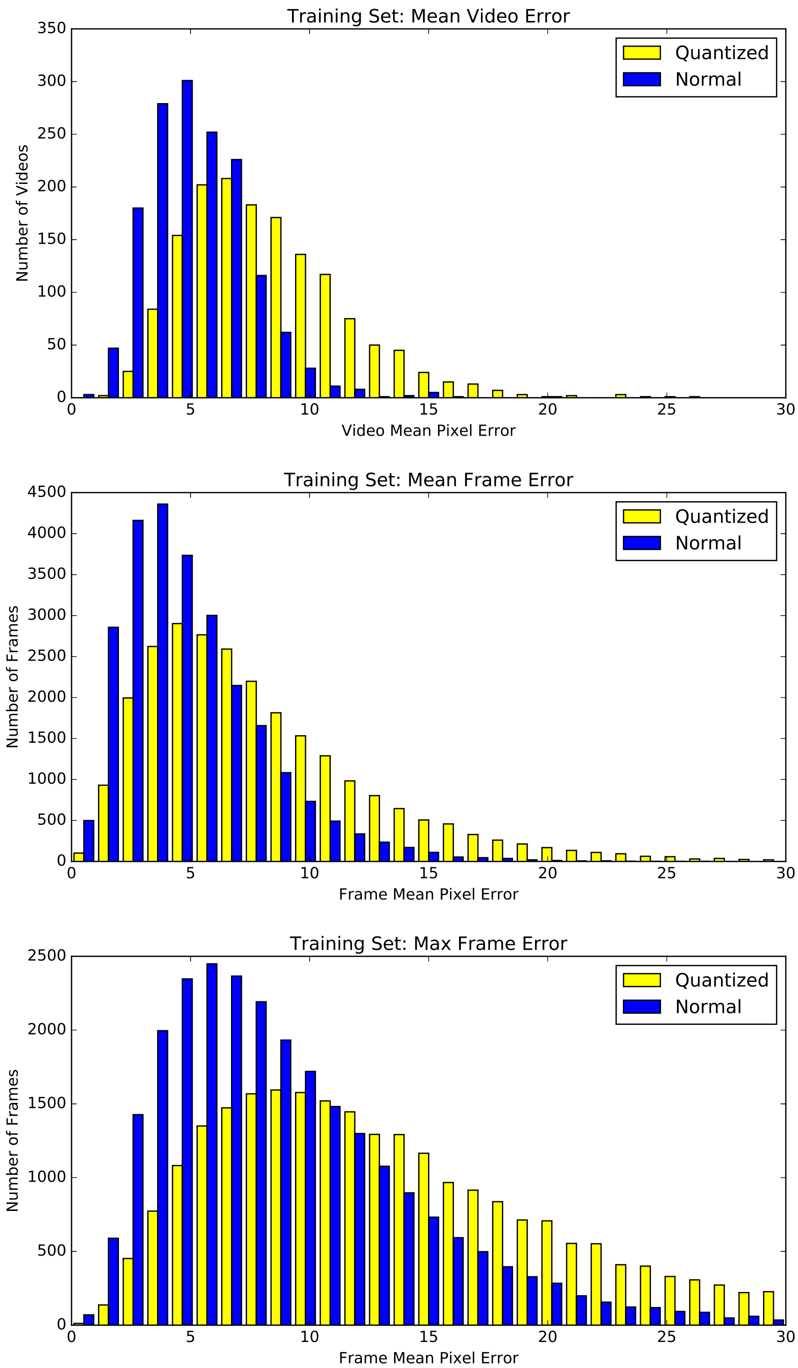

Figure 13: The Inception struggles in general, but shows significant losses with quantization

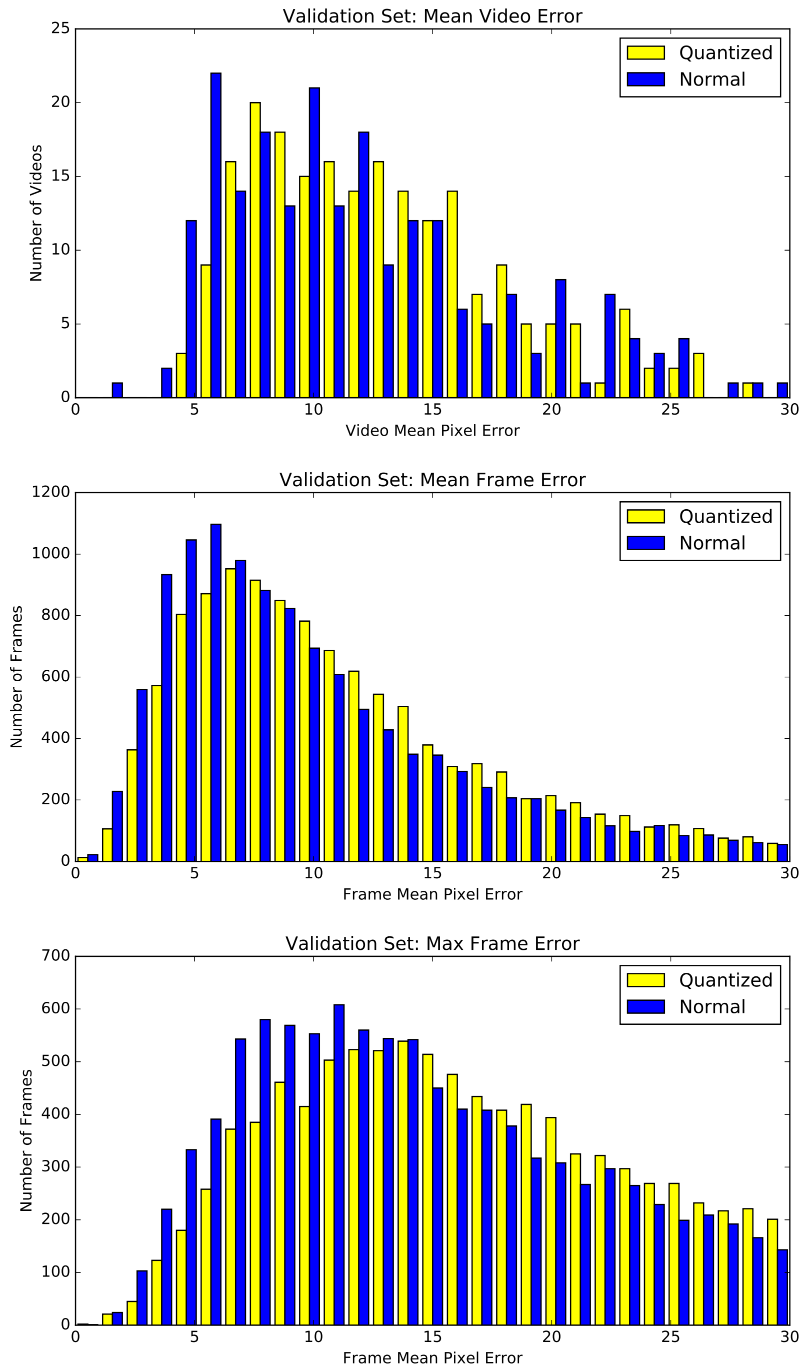

Figure 14: The Inception validation encores the training set, with considerable losses after quantization

5 Conclusion

Using CaffeNet, this thesis has successfully demonstrated that it is possible to quantize an object tracking network. This compression process verified the expected 75% reduction in network size. However, it is the minimal change in accuracy that is particularly impressive.

On accuracy metrics, the CaffeNet network after quantization demonstrated effectively the same accuracy as it had prior to quantization. Only a minor shift from the number of pairs with a mean error less than one to an mean error between 1-2 is of note. This however is still an extremely good result.

It can therefore be concluded that quantization is definetly something that should be considered for object tracking networks, even when they are not being employed for embedded devices. The potential tradeoff is simply too good to ignore; a significant reduction in storage requirement and memory requirement, along with the potential for considerable performance gains that have previously been demonstrated [SH16][Har].

The second network, Inception, unfortunately struggled with achieving satisfactory accuracy. However, these issues however stem not from the quantization process, but rather from the network struggling to learn the original problem. This results in a network that does not make for a very suitable tracker. While the quantization results are not great for Inception, because it was never able to learn the original problem properly, they are not very meaningful.

5.1 Further Work

Quantization is not the only technique available for reducing network size. The current CaffeNet network has quite a large number of parameters in the fully connected layers, but there may be many that are minimally contributing to generating any results.

Another interesting compression technique that could be investigated is pruning. Pruning allows for the removal of weights that are contributing minimally to the network output. By removing them, the number of computations that the network needs to perform are reduced. Obviously, their removal also results in a further reduction in the general size of the network. This has demonstrated good results in other applications[SH16].

References

[Bö] Lars-Christer Böiers. *Lectures on Optimization.*

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[Don] Jeff Donahue. Caffenet.

[Eth] Ethereon. Caffenet tensorflow converter. [Online; accessed 24-November-2016].

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[Har] Mark Harris. New pascal gpus accelerate inference in the data center.

[HTS16] D. Held, S. Thrun, and S. Savarese. Learning to Track at 100 FPS with Deep Regression Networks. *ArXiv e-prints*, April 2016.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[Inc16] Google Inc. Tensorflow. 2016. [Online; accessed 24-November-2016].

[KML+15] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. ˇ Cehovin, G. Fernández, T. Vojíˇ r, G. Häger, G. Nebehay, R. Pflugfelder, A. Gupta, A. Bibi, A. Lukežič, A. Garcia-Martin, A. Petrosino, A. Saffari, A.S. Montero, A. Varfolomieiev, A. Baskurt, B. Zhao, B. Ghanem, B. Martinez, B. Lee, B. Han, C. Wang, C. Garcia, C. Zhang, C. Schmid, D. Tao, D. Kim, D. Huang, D. Prokhorov, D. Du, D.-Y. Yeung, E. Ribeiro, F.S. Khan, F. Porikli, F. Bunyak, G. Zhu, G. Seetharaman, H. Kieritz, H.T. Yau, H. Li, H. Qi, H. Bischof, H. Possegger, H. Lee, H. Nam, I. Bogun, J.-C. Jeong, J.-I. Cho, J.-Y. Lee, J. Zhu, J. Shi, J. Li, J. Jia, J. Feng, J. Gao, J.Y. Choi, J.-W. Kim, J. Lang, J.M. Martinez, J. Choi, J. Xing, K. Xue, K. Palaniappan, K. Lebeda, K. Alahari, K. Gao, K. Yun, K.H. Wong, L. Luo, L. Ma, L. Ke, L. Wen, L. Bertinetto, M. Pootschi, M. Maresca, M. Danelljan, M. Wen, M. Zhang, M. Arens, M. Valstar, M. Tang, M.-C. Chang, M.H. Khan, N. Fan, N. Wang, O. Mikš´ ık, P. Torr, Q. Wang, R. Martin-Nieto, R. Pelapur, R. Bowden, R. Laganière, S. Moujtahid, S. Hare, S. Hadfield, S. Lyu, S. Li, S.-C. Zhu, S. Becker, S. Duffner, S.L. Hicks, S. Golodetz, S. Choi, T. Wu, T. Mauthner, T. Pridmore, W. Hu, W. Hübner, X. Wang, X. Li, X. Shi, X. Zhao, X. Mei, Y. Shizeng, Y. Hua, Y. Li, Y. Lu, Y. Li, Z. Chen, Z. Huang, Z. Chen, Z. Zhang, Z. He, and Z. Hong. The visual object tracking vot2015 challenge results. In *ICCV workshop on Visual Object Tracking Challenge*, pages 564 – 586, December 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probalistic Perspective*. Massachusetts Institute of Technology, 1 edition, 2012.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[SH16] Robin Seibold and Björgvinsdóttir Hanna. Face recognition based on embedded systems. 2016.

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[War16] Pete Warden. How to quantize neural networks with tensorflow. 2016. [Online; accessed 24-November-2016].

Master's Theses in Mathematical Sciences 2016:E53
ISSN 1404-6342

LUTFMA-3309-2016

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/