UPPSALA
UNIVERSITET

IT 23 108

Degree project 30 credits

November 2023

Methods for Developing Tiny Convolutional Neural Networks for Deployment on Embedded Systems

Tobii AB

Egemen Yiğit Kömürcü

Master's Programme in Computer Science

UPPSALA
UNIVERSITET

UPPSALA
UNIVERSITET

Methods for Developing Tiny Convolutional Neural Networks for Deployment on Embedded Systems

Egemen Yiğit Kömürcü

Abstract

With the recent development in the Deep Learning area, computationally heavy tasks like object detection in images have become easier to compute and take less time to execute with powerful GPUs. Also, when employing sufficiently larger models, these daily tasks are predicted with greater accuracy. However, these models are not widely used since the embedded devices have computational and memory limitations. Therefore, this project aims to explore different methods to compress these large networks into smaller ones. In that way, lighter models can be deployed on embedded devices without losing too much accuracy from the large models. Firstly, this work is started by training the baseline model ResNet50 which is available in Tobii´s Deep Learning framework developed for their eye-tracking glasses and devices. Then, different lighter modern architectures like Resnet18, MobileNetv2, and SqueezeNet are attempted to be trained to get reasonable accuracy with less model complexity. Three model compression techniques will be applied in this thesis. The first one is a method called distillation which is conducted to increase the accuracy of the smaller networks. The second step is to experiment with various pruning methods like threshold, layer, and channel pruning to decrease the model size of the SqueezeNet model. Since the quantized Squeezenet model is not available in Pytorch, the quantization method from 32-bit to 8-bit is applied to the model to further decrease the model complexity. Finally, in this thesis it is found that it is possible to reduce the baseline model of 100 MBs with an overall loss (0.77) less than 1, into the layer pruned quantized SqueezeNet of less than 1 MBs which has an overall loss of 1.44 which does not exceed 1.5 pixels.

Faculty of Science and Technology

Uppsala University, Uppsala

Supervisor: Alexander Davies & Oscar Danielsson - Tobii AB

Reviewer: Stefanos Kaxiras

Examiner: Mats Daniels

UPPSALA
UNIVERSITET

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

AutoML Automated Machine Learning

CNN Convolutional Neural Network

conv convolutional

DL Deep Learning

DNN Deep Neural Network

GDPR General Data Protection Regulation

NAS Neural Architechture Search

ONNX Open Neural Network Exchange

ResNet Residual Networks

RL Reinforcement Learning

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are increasingly being used in several industrial computer vision applications due to their superior performance compared to previous techniques like traditional Machine Learning methods. There have been a plethora of competitions going on for image classification tasks like object detection from ImageNet datasets provided by Google. Researchers have found ways to explore new deep-learning models that predict these tasks with fairly good accuracy. In addition to the classification tasks, there exist other application areas in deep learning. For example, gaze estimation can be used in eye-tracking technology which is a regression task.

However, mainstream CNN architectures are still too large and computationally expensive to run on embedded systems, prohibiting widespread usage. Several techniques have been developed to reduce the memory footprint and computational costs of CNNs including Pruning, Distillation, Quantisation, and Neural Architechture Search (NAS) - all to varying levels of success. Unfortunately, these methods have yet to produce CNNs with comparable memory and computational requirements as existing methods optimized for embedded systems. The goal of this project is to perform the aforementioned compression techniques on various network architectures to produce CNN models for eye tracking tasks on synthetic eye image datasets so that these small deep learning models can be deployed on the embedded system specification provided by Tobii [1].

1.1 Research Questions

In this project, several model compression techniques are to be implemented and evaluated to ascertain each technique's suitability for use on embedded

systems. To avoid the need for premature deployment on embedded systems, the profiling of the CNNs and the traditional method is done on the development computer. The size and complexity of the CNN are evaluated by recording the following:

1. The accuracy of the model
2. The size and complexity of the model:
    - The model size (in MB)
    - The number of parameters
    - The time required for a single forward pass (inference or execution time in seconds)

In this work, eye tracking is used to predict pupil and glint features of the eye. Eye tracking is a multi-object detection and regression task. The aim is to find a CNN having a size smaller than 1 MB which is significantly smaller than the baseline network. This should be carried out without losing too much accuracy from the baseline model. The baseline model is selected from Residual Networks (ResNet) [2], which has a layer size that can vary from 18 to 152. The ResNet models are provided by a Deep Learning (DL) framework from Tobii. The first step of this project is to experiment on various network architectures like MobileNetv2 [3] and then examine the three traditional model compression methods listed as distillation, pruning, and quantization and report the findings of each method so that the results can benefit future research within this area. A literature review will be conducted to investigate how new but smaller models can be obtained with the help of new approaches such as NAS [4] [5] by using either Automated Machine Learning (AutoML) [6] or Transformers [7] models.

1.2 Research Methodology

During this work, a training framework written using the PyTorch neural network library is provided by Tobii. The training code has already been tested for training a neural network and providing relevant training statistics. Within this framework, various compression techniques are implemented and evaluated. As well as ensuring that the CNN successfully fulfills the hardware requirements of the embedded system, the performance of the signal should also be studied to understand the trade-off between model compression and signal

degradation better. Various signal performance metrics of the CNNs will be analyzed. The main metric to be analyzed is the overall loss which is calculated as the difference between the CNN output and the ground truth on an evaluation dataset. These metrics are part of the training framework and can be readily viewed using the TensorBoard tool. The training framework also has support for exporting models to a format suitable for inference in products. For the inputs, provided synthetic datasets for training and evaluation are downloaded to conduct experiments. There is also a separate framework for generating additional synthetic data if required. For deployment on the development machine, there is support for exporting trained models to a readable format using Open Neural Network Exchange (ONNX). Lastly, the results and overall progress throughout the project are documented via Confluence and JIRA.

1.3 Delimitations

This thesis aims to implement and evaluate different deep learning model compression techniques to obtain the smallest model without losing too much accuracy in comparison to the large baseline model. For this reason, it is not expected that the trained CNN will be the optimal trained model for the given dataset nor is the results expected to be generalized to other evaluation datasets. Datasets will only contain synthetic images and not images of real scenes. Although desirable, it is not expected that the prototype should run on live data. The prototype is only required to run on the designated datasets. Due to time constraints, deployment of the solution on the target embedded system is skipped. However, comparable profiling and evaluation data between the solution and the traditional method on the development computer is obtained. Ideally, the evaluation metric values are to be compared against the size and execution of the existing conventional method used in the embedded system which is an eye-tracking device. However, the already deployed model is not available to compare with the deep learning models in the existing framework.

1.4 Social Impact and Sustainability

The consequences of this project may have significant societal impacts on the application areas of the eye-tracking device. One example of an area where the eye tracking device is applied is the virtual reality device called Tobii Pro which can enhance the experience of drivers by assisting and warning the

driver whenever there is an approaching danger. Another example of using these devices is within research in children and infantry where the child's interaction with their environment and people is studied, to aid parents in their nurturing.

Moreover, Tobii Dynavox, a subsidiary branch within Tobii [8] produces specialized products for people with disabilities to facilitate them to use computers without having to use a keyboard and mouse. These devices make use of lightweight and accurate deep-learning models, and developments within this field may facilitate a greater availability of these devices. Hence, the findings of this thesis may have a positive effect on equal education quality, leading to students with disabilities not being disadvantaged compared to other students. In addition to societal impacts, the prevalent usage of these embedded devices can be beneficial to Tobii's Sustainable Development Goal (SDG) [1]. As the eye-tracking models become lighter and thus require less storage memory, the tasks can be completed with decreased inference time. Therefore, improving these technologies can help achieve the goals of saving cost and becoming energy efficient.

1.5 Ethical considerations

This study has social and sustainability impacts as benefits since it proposes cheaper and more prominent use of eye-tracking devices. However, possible negative effects are that the findings of this study may be used for unethical purposes. For instance, lightweight accurate models for eye tracking may make users' eye data more available for organizations who have the intention to profile users to analyze their behaviors and the characteristics of their eyes. Furthermore, these datasets containing user data can be leveraged in other projects regarding eye recognition with malicious intentions which would be considered a data breach and would cause severe harm to the user's privacy. In these cases, it is vital to make use of General Data Protection Regulation (GDPR) and to inform the users on how their data is intended to be used.

1.6 Structure of the thesis

- Chapter 1 introduces the project, the purpose, and the goals of the work.
- Chapter 2 presents relevant background information about CNN, modern network architectures, and model compression methods like pruning, quantization distillation, and NAS.

- Chapter 3 describes the dataset characteristics, preprocessing, and implementation methodology of the model compression techniques used to solve the problem and achieve the target.
- The outcome of the implementation of the methodology is described in Chapter 4 by including the experiments and their results. It also presents the analysis of the conducted experiments.
- Chapter 5 and 6 discuss the possible outcomes of the results of experiments and interpretation of them.
- Chapter 7 includes suggestions for future work and can act as guidance to the readers who will endeavor to reach similar goals.

Chapter 2

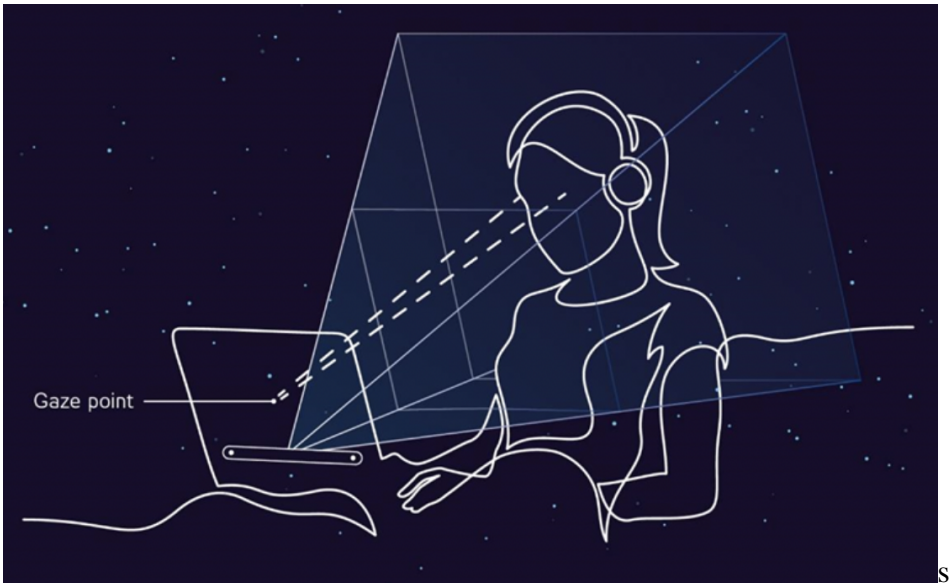
Background

In the 1950s Hebb coined the idea of Neural Networks which mimics the human learning algorithm i.e. neurons send information to other neurons within the brain. This idea was introduced as a subfield within Artificial Intelligence called Machine Learning. Hebb described ML as the computer's ability to learn without being explicitly programmed to do so. [9]

In the coming parts of the background, there is a division into 4 sections. Firstly, the eye-tracking task is described to introduce the reader to the subject of this thesis. Secondly, the next section compares the differences between the fully connected layers in deep neural networks and convolutional layers in CNN. Then, various examples of modern network architectures are given as well as a description of how they work. Lastly, the theoretical aspects of the model compression methods are presented. The model compression methods to be introduced are pruning, distillation, quantization, and NAS.

2.1 Eye-tracking task

Eye tracking i.e. gaze estimation is an object detection task combined with the regression task which when combined predicts where the pupil and glints of the eye are located to predict where the eye gaze is looking at a screen.



s

Figure 2.1: Eye tracking gaze

Eye tracking components typically include one or more high-resolution cameras, near-infrared light sources, and algorithms that translate head and eye movements into real-time data streams, leveraging machine learning and advanced image processing. [10]

The eye-tracking process flow is as the following: [10]

1. An eye tracker consists of cameras, a projector, and algorithms.
2. The light sources create reflections — called glints — on the cornea of each eye and the projectors create the pattern of the infrared lights on the eyes.
3. At a high frame rate, the cameras take high-resolution images, recording the person's eyes and the glints.
4. The algorithms use machine-learning techniques to identify the pupils and the glints.

5. Using advanced image processing and knowledge of human eye anatomy the algorithms can calculate data points, such as pupil position, gaze vector of each eye, gaze point, and eye openness.

The pupil is the black circle in the eye and the glints are the small circles located on the sides of the main circle. Detecting whether a circle in an image is the pupil, the glints or neither is the image detection task whilst predicting the localization of these circles is the image regression task.

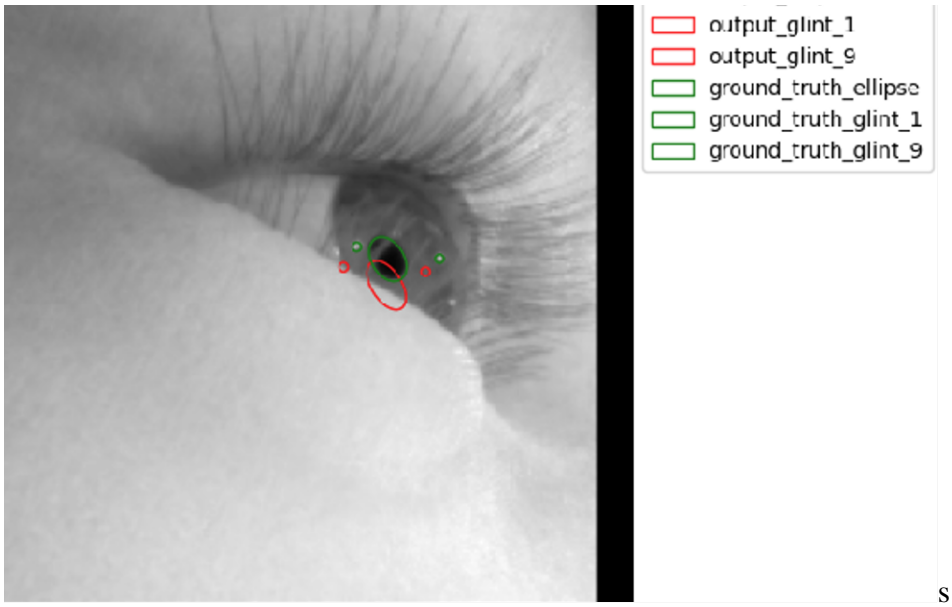


Figure 2.2: Eye pupil prediction vs. actual location

In Tobii's DL framework, an overall loss is implemented as it is the linear combination of the average pupil and glint position error.

2.2 Deep Learning

2.2.1 Deep Neural Networks

Fully Connected Layers Artificial or deep neural networks are made up of fully connected layers that compute every input value with every weight scalar. Neurons in a fully connected layer have connections to all nodes in the previous layer. This produces a result in which all combinations of neuron values and the weights are considered thus some of them may be irrelevant, for instance, if the weights are very close to zero. Then the values from previous layers do not need to be used.

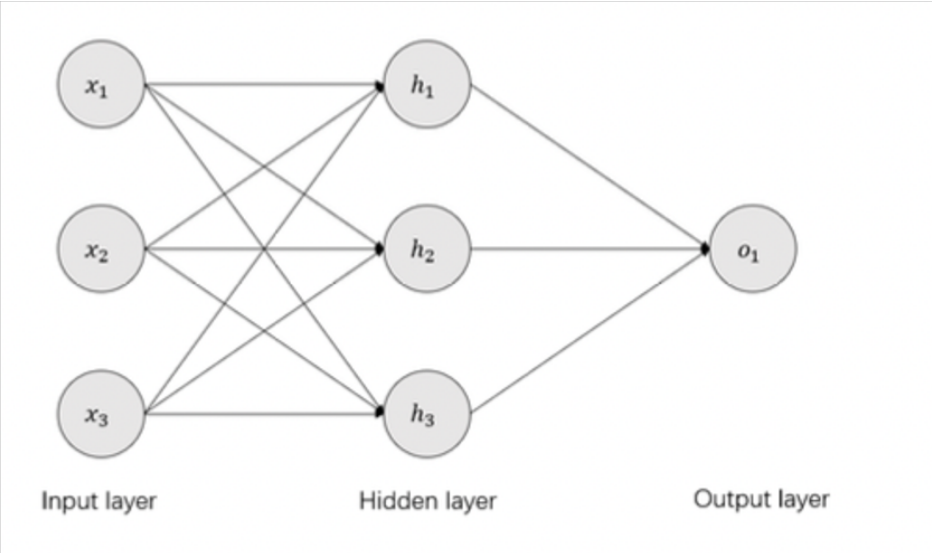


Figure 2.3: Fully connected layers

A basic Deep Neural Network (DNN) can include the following components:

- Input layer: it takes the input and propagates the prediction to the hidden layers.
- Weight, bias: in each layer, the predictions are calculated by the combination of the weights, bias, and input data. The loss function of the network then tries to find the best weights and biases for each layer by applying back-propagation.

- Hidden layers: these layers take the outputs from the input layer, calculate the prediction by its weights and bias, and propagate the output to the last output layer.
- Activation functions: these functions cast the output value to zero since the output can not be negative. If the task is an image classification task that requires labeled integer output, the Sigmoid activation function is used to label the output prediction. If the task is an image regression task that can have a decimal number, the ReLU activation function is used to pass the output the same unless it is negative.
- Output layer: it applies the output from the last hidden layer to the activation function and sums up the prediction of the input data as output in the last step in the end-to-end process.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are inside the subset of DNN or Deep Learning algorithms which are mostly used for solving image prediction problems. These tasks can include image classification and image regression tasks as well as synthesizing images. Different low-level features of the images can be learned via CNN architectures. [11]

Convolutional Layers CNN architectures differ from DNN by having convolutional layers. These layers have a filter called kernels and the predictions of the image pixels are calculated by the convolution multiplication between input image pixels and this kernel. Then these filters slide over the image and sum the the element-wise multiplication results. This convolution operation can be used to detect features of the images like edges, texture, patterns, etc. inside each layer. It is highly hard to explain the output of each layer so it is really hard to choose which layer is important or unnecessary to have.

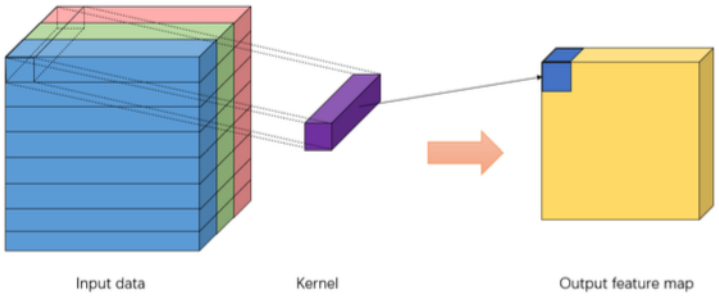


Figure 2.4: Convolutional layers

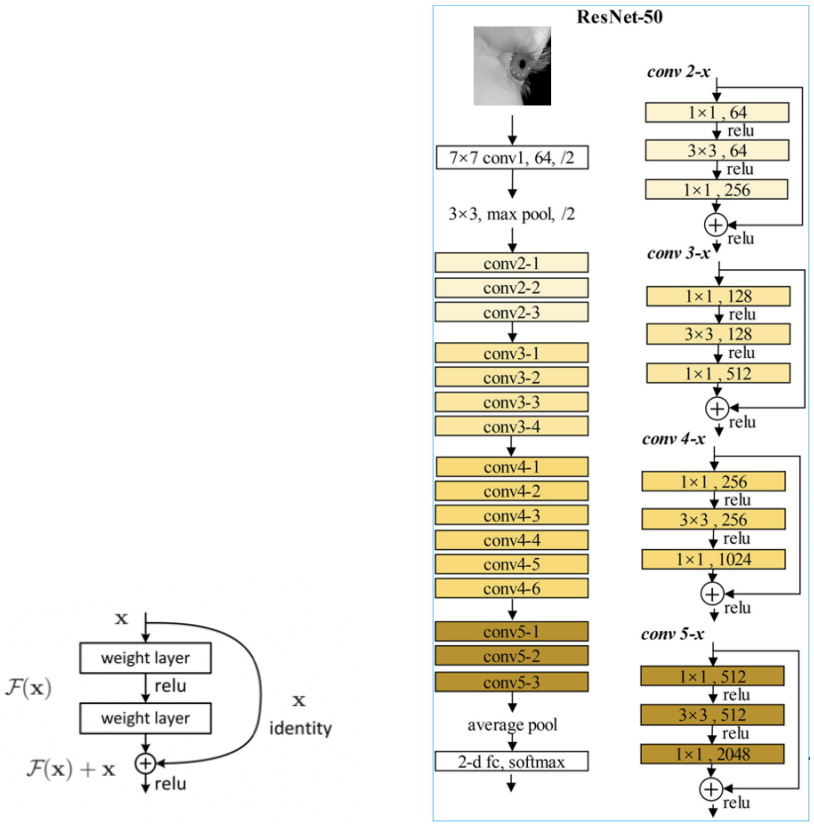
Convolutional layers produce features that both reduce the input size and enable specific groups of values to be used multiple times. Thus it decreases all the possible combinations to be used in each layer. It is still over-parameterized and researchers are exploring ways to decrease the number of convolutions. [12]

Pooling Layers Another different layer to be used CNNs is pooling layers. These layers are used to reduce the dimensions like the width and height of the images. however, it does not reduce the accuracy much. Thus pooling reduces the model size and number of parameters in the model. These layers are mostly applied after applying convolutional and activation layers in CNN architectures. The most common pooling techniques are max pooling and average pooling techniques. [12]

2.2.3 Modern network architectures

Several modern architectures have been introduced that can process various common tasks. These common tasks include object detection, segmentation, classification, and regression. Recently, there have been competitions on object detection for image classification tasks on public data such as CIFAR, COCO, and ImageNet datasets provided by Google. To achieve good accuracy with image detection which can detect more than 250 outputs including, for instance, eyes, humans, cows, or birds, researchers have produced various network architectures such as ResNet, MobileNet, InceptionNet, GoogleNet, and SqueezeNet. These architectures aim to achieve the best accuracy for Top-1 and Top-5 losses for the aforementioned datasets. Top-1 means the average error between the actual class and the most probable predicted class whilst Top-5 is the average error between the actual class and the five most probable predicted classes. Each of the network architecture´s accuracies for ImageNet datasets are presented in Appendix A.1.

ResNet Residual Networks which are commonly referred to as ResNet are inspired by the following phenomenon. The model takes the outputs of the previous layers which are fed into the latter layers to preserve the accuracy and lose the information which is gained at previous layers. [2] [13]



(a) ResNet block prediction

(b) ResNet whole model prediction

Figure 2.5: Residual Networks

The residual blocks eliminate the vanishing gradient problem and make the accuracy constantly increase as the number of residual blocks increase, i.e. it avoids over-fitting. At the end of the model backbone, the size of the output feature map becomes 2048.

Mobile-Net MobileNet is a model that uses depth-wise separable convolutions. A depth-wise separable convolution is a factorized form of a full convolution, which first applies a single kernel to the input (depthwise) and then applies a 1x1 convolution (pointwise) to combine the outputs of the depthwise operation. [14], [3]

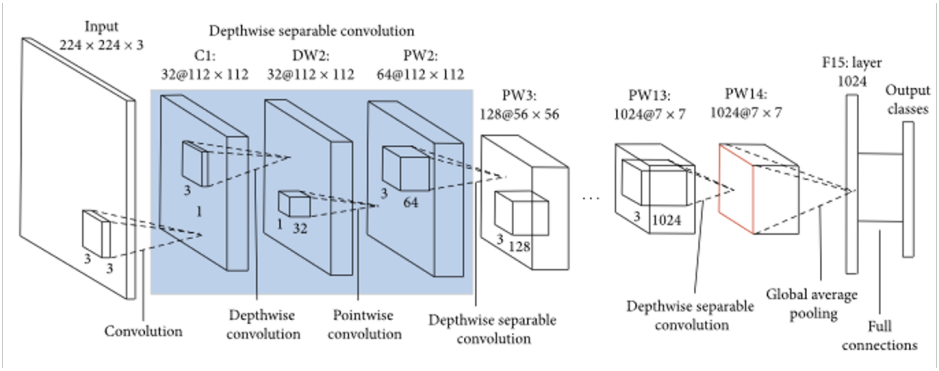


Figure 2.6: MobileNet depthwise convolution

It significantly reduces the number of parameters when compared to the network with regular convolutions with the same depth of the nets. This results in lightweight deep neural networks. At the end of the model backbone, the size of the output feature map becomes 512.

ShuffleNet ShuffleNet promises a lightweight model architecture without decrementing the accuracy too much. [15]

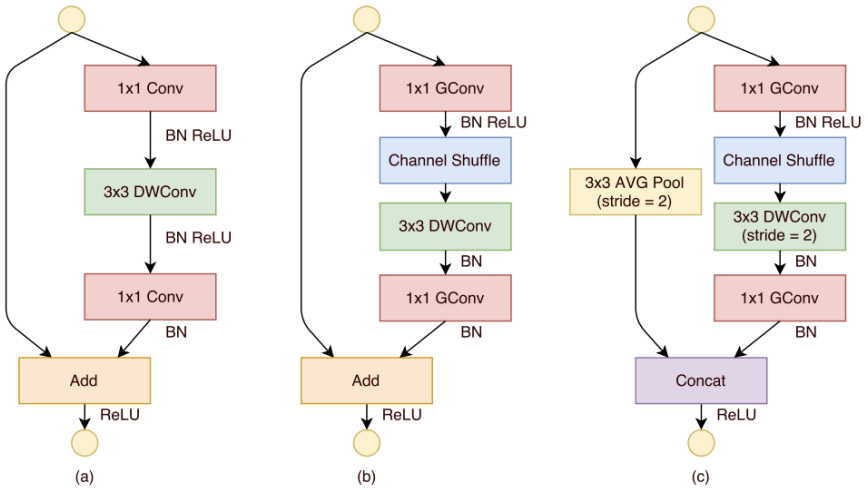


Figure 2.7: ShuffleNet

EfficientNet The idea behind EfficientNet is to scale the output of the different sub-networks and combine them. [16]

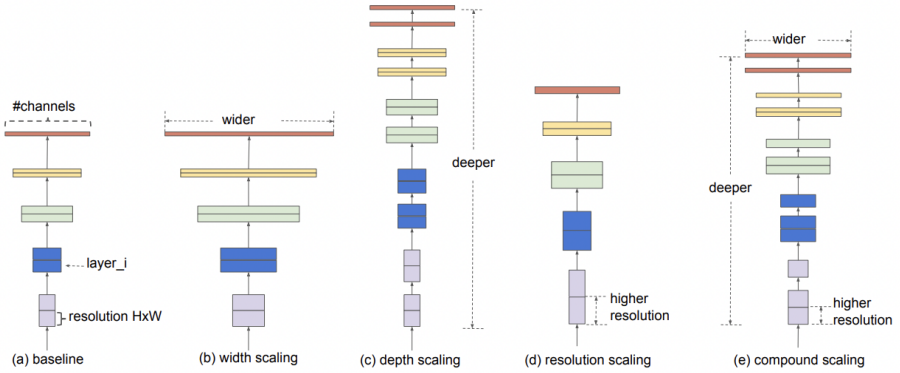
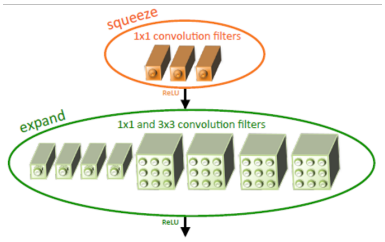
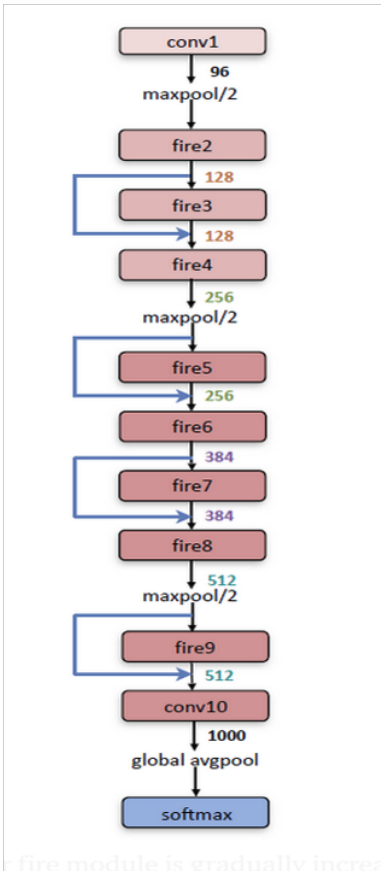


Figure 2.8: EfficientNet

Squeeze Net The third promising architecture is SqueezeNet, developed in 2019. It is the compressed version of AlexNet with the same accuracy for open source datasets like Imagenet, PASCAL, etc. [17]



(a) Fire module prediction



(b) Squeezenet model prediction

Figure 2.9: SqueezeNet

As shown in the model above and to the left, the model has 9 fire modules as blocks, each consisting of squeeze and expand layers.

As its name suggests, 'squeeze' layers decrease the input size of the layer into the specified output size whereas the 'expand' layers increase the image sizes. For example, the first fire module gets the output of the first convolutional (conv) layer which is 64, and inside the fire module, the squeeze layer applies conv to shrink into a 16-size image. Each expand layer takes this 16-size image and expands it to an image size of 64 with 1x1 and the other 3x3 conv filter. Then these two outputs are combined so that the input of the fire module becomes 128 from 64. At the end of the model backbone, the size of the output feature map becomes 512. The more detailed SqueezeNet layer-by-layer architecture can be found in Appendix A.2 and Appendix A.3

2.3 Model Compression Techniques

This section presents various commonly used model compression techniques. Firstly, a technique called pruning is introduced. Furthermore, distillation and quantization techniques are introduced. The last section is dedicated to describing NAS.

2.3.1 Pruning

The model compression technique pruning includes two types of pruning: unstructured and structured pruning.

Unstructured Pruning

In unstructured pruning, individual weights depicted as synapses and neurons below, are removed or zeroed one by one. In this way, the outcome could be that only one neuron is removed in one layer and multiple neurons are removed in another layer. Hence, it is referred to as unstructured pruning. Unstructured pruning can potentially improve the accuracy but such implementation can become complex.

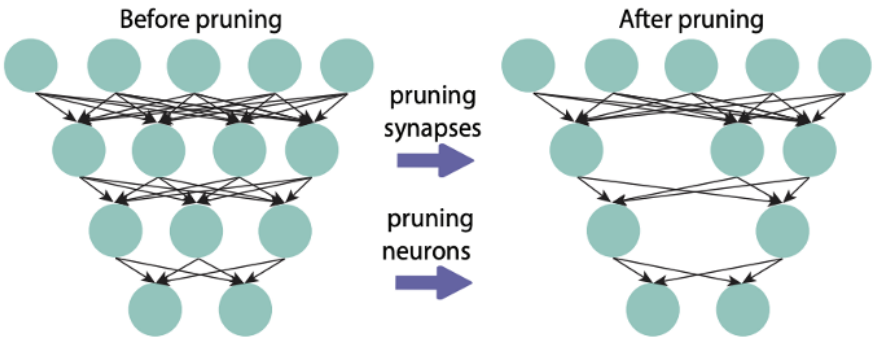


Figure 2.10: Unstructured pruning

Structured Pruning

In contrast to unstructured pruning, in structured pruning, the common approach is that either the filter, the channel, or some filter & shape combination is removed. In that way, removing the weights in the whole layer or channel is needed so that the model can be reduced. This may decrease the accuracy as removing a whole filter may result in removing the crucial weights.

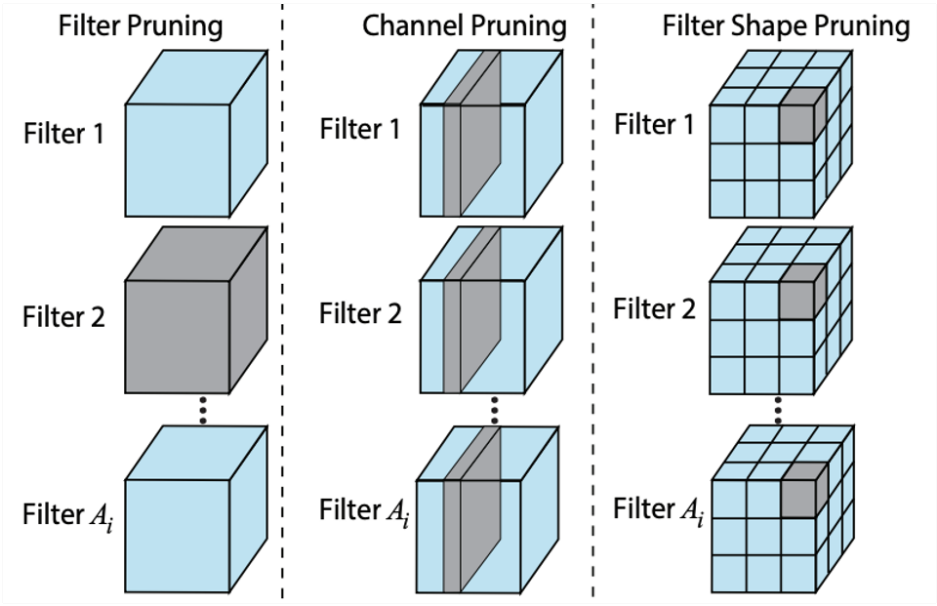


Figure 2.11: Structured pruning

2.3.2 Distillation

The second type of model compression method is knowledge distillation. In knowledge distillation, the idea is that there exist two types of networks: a teacher network and a student network. The teacher network is a large-sized model that can obtain good accuracy after default training. The student network is a much smaller network that can learn the pattern of the images by using the ground truth of the images as well as the soft prediction generated by the teacher network.

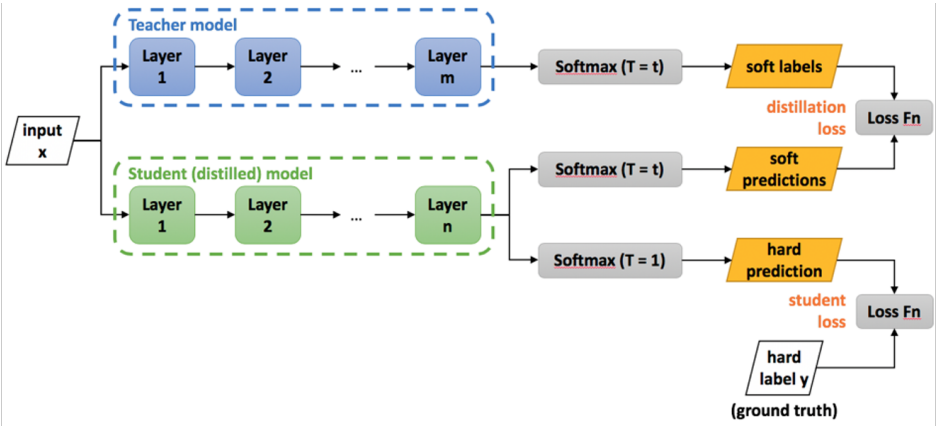


Figure 2.12: Knowledge distillation flow

The actual loss is calculated using the difference between the student prediction and the actual outputs, while the distillation loss is the difference between the student and teacher predictions. The total loss is based on the linear combination of these two losses.

In image classification tasks, after the backbone, there exists a softmax layer that casts the output value into the probability of the predicted class.

$$p = \frac{\exp z_i/T}{\sum_j \exp z_j/T} = softmax(\frac{z}{T}) \tag{2.1}$$

Since the actual loss is of importance, the distillation loss can be diminished by setting the temperature to a higher value which makes it soft loss whilst the actual loss temperature is set as 1.

2.3.3 Quantization

Furthermore, there is a method called quantization. In image quantization, the size of an image file can be compressed by for instance reducing the number of colors or decreasing the changes of the contrast in an image.

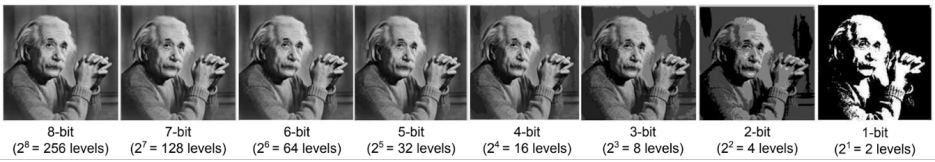


Figure 2.13: Quantization of image

The figure above depicts how the bit size of an image can be altered by using quantization. The leftmost Einstein image is represented in gray-scale 8-bit values. As contrast is increased in each image the bit representation is decreased from 8 bits to 1 bit. Figure 2.14 illustrates an example that an image can lose information and quality but remain interpretable.

The quantization is conducted by assigning a value close to the actual value. For instance, if the value's floating point is 200.2 it will be cast into 200 and if it is 200.6, it will be cast into 201.

This project also includes applying quantization to the weights where the weights are the 8-bit gray-scale values followed by 32-bit floating points found in the layers represented as matrices. Rather than quantizing 8 bits, weight quantization mainly focuses on removing the numbers after the floating point or assigning them via a logic that casts them into the closest number.

One of the quantization approaches is formulated as below :

$$Q(x, scale, zero\_point) = round(\frac{x}{scale} + zero\_point) \tag{2.2}$$

2.3.4 Neural Architecture Search

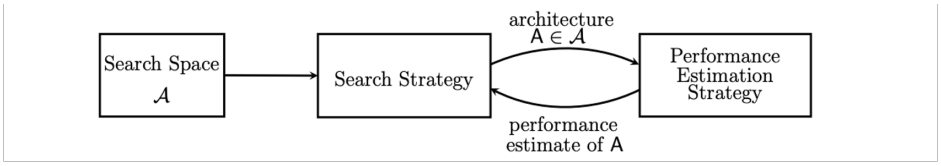


Figure 2.14: Neural Architecture Search flow

There exists a vast of possible network architectures that can solve the machine learning problems. That requires much experimentation to find the optimal neural networks for the specific need. So Neural Architecture Search (NAS) comes in to find the optimal network architecture which automates the design of these networks without needing much experimentation.

In NAS, there exist possible network architectures having different numbers of layers, nodes, and connections between nodes in the search space. In the search strategy, there is a guide to explore what determines to be searched for in the next network via the strategy algorithm. The most prominent algorithm approaches are by using Reinforcement Learning (RL), random wise, etc. Then, the performance of this network is estimated and serves as feedback on the search strategy to improve the NAS. [4] [5]

DeepCompression DeepCompression is a model compression pipeline that involves several compression techniques and steps. It is a three-stage pipeline that includes pruning, trained quantization, and Huffman encoding respectively. [18]. Deep-compressed SqueezeNet is one of the promising networks that is achieved by using this deep compression pipeline and it could reduce the model size from 4.8 to 0.47 MB without losing much accuracy. [17]

AutoML Solving a machine learning problem requires several steps from end to end after taking the input. Such steps include data preprocessing, feature engineering, finding the model, hyperparameter tuning, inference, deployment, etc. If these steps could be considered a pipeline, Automated Machine Learning (AutoML) is the automation technique that tries to find the optimal pipeline end-to-end without needing the developers to experiment with each stage in the pipeline. For example, AutoML removes the need for the person to apply NAS to find the best model. So NAS is a subset of AutoML since selecting the model is one of the steps included in the pipeline.

Chapter 3

Methodology

3.1 Dataset

The synthetic datasets are downloaded by using a script that is available in the framework. The dataset names are represented as XRYS_image_id where the image id is a 4-digit number. There also exists an image prefix to be set to download more images in the downloader script. In addition to that, the number of images per recording can be set to download larger datasets. For instance, the size of the synthetic datasets can be varied with the aforementioned parameters as the table below represents.

The dataset which is downloaded by setting the dataset name as 1034 is not of a sufficient size to be used for training. To converge a model better a larger dataset is necessary. For that purpose, after tweaking the number of images per recording to 32, the dataset size becomes 2 GB. It can be increased up to 6.6 GB if the dataset name is modified into 1 which also includes the two former datasets.

Image dataset name	Number of images per recording	Total Dataset size (GB)
1034	1	0.072
10	32	2.0
1	32	6.6
2-9	32	1.8

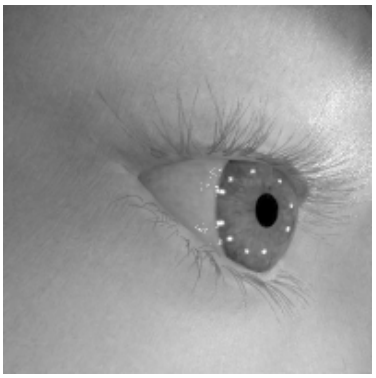
Table 3.1: Downloading total training and validation datasets for different image prefixes

Datasets	Ear		Nose		Total
	Left(0)	Right(2)	Left(1)	Right(3)	
Training	36448	36448	36448	36448	145792
Evaluation	29184	29184	29120	29184	116672
Total	65632	65632	65568	65632	262464

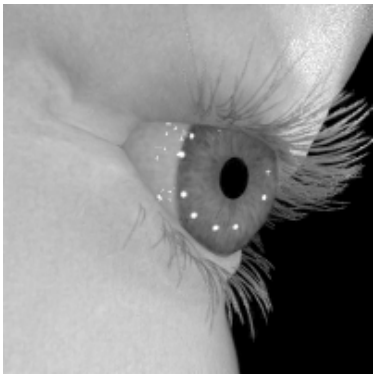
Table 3.2: Number of images for different image recordings for image prefix of 1

There are approximately 73k and 59k images in the training and evaluation datasets for ear and nose recordings. The input images are 224 x 224 grayscale images which are converted into RGB type in the PyTorch framework. The sample images for training and evaluation datasets are shown below:

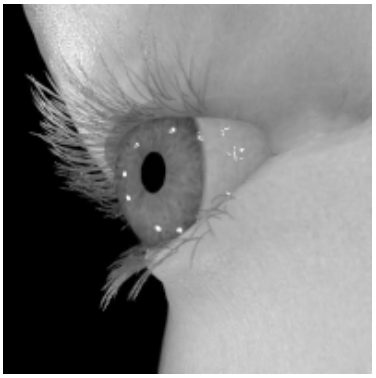
(a) Left Ear Recording: 0



(b) Right Ear Recording: 2



(c) Left Nose Recording: 1



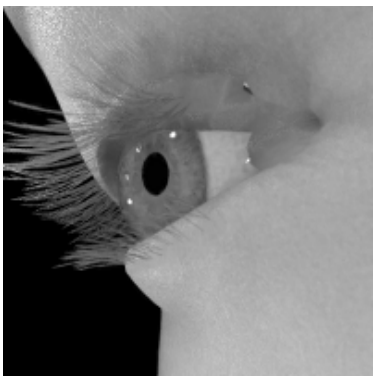
(d) Right Nose Recording: 3

Figure 3.1: Illustration of various sample image recordings for training set

(a) Left Ear Recording: 0

(b) Right Ear Recording: 2

(c) Left Nose Recording: 1



(d) Right Nose Recording: 3

Figure 3.2: Illustration of various sample image recordings for evaluation set

3.2 Preprocessing

The original images are in the format of 200x200. However, the framework implementation requires that input images to the model should be 224x224. Thus, a padding frame having 12 pixels is applied from each side to make the images 224x224. Furthermore, these images are also in gray-scale format which has the values on only one channel of the image. Since the framework implementation also requires the input to have 3 channels, the gray-scale images are transformed into RGB images which have 3 channels. Consequently, the padding layer and gray-scale to RGB layer form preprocessing layers for each model in the framework.

3.3 Baseline

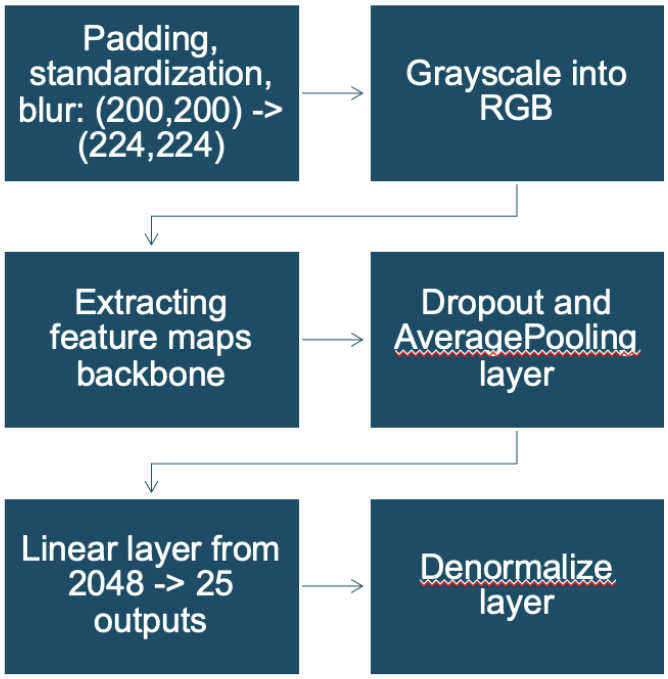


Figure 3.3: Baseline model flow

Since the synthetic input images are represented in grayscale, the input data has only one channel having a width and height size of 200, 200. Padding is applied in Tobii´s DL framework to match the input size into 224, and 224. Then the input values in one channel are augmented into 3 channels containing the same values. This is necessary since the existing networks in Pytorch expect 3 channels as inputs.

The baseline model is chosen as Resnet architecture. The number of layers of Resnet are 152, 101, 50, 34, and 18 and they are already implemented and ready to be experimented with in Tobii´s DL framework in Pytorch. Upon experimenting with Resnet of 50 layers and obtaining a good result having a loss of less than 1, it is chosen as the baseline model. Another reason why Resnet of 50 layers is chosen is that the model has a size of 100 Mbs, which will be a reasonable size and can be put to comparison with Resnet152 where the size amounts to 240 Mbs as depicted in Appendix A.1 [2] [13]. Rather than using the classifier layers which are default in the torchvision.models package

in Pytorch, the existing framework uses only the feature map of the networks in order not to use the weights for general object detection tasks for ImageNet datasets. The feature maps are then connected to the final block of Average Pooling and Dropout layers. The former layer is being used for downsampling the feature maps by averaging multiple pixels whilst the latter one reduces the probability of overfitting by zeroing 20 percent of the matrix indices of the outputs.

When the baseline model is chosen, it is implemented to be used as the backbone of the model. If the prediction task was a classification task it would have required a softmax layer at the end. The eye tracking task is an image multi-regression task predicting the location of the pupil and glints of the eye. This instead requires the model to have a final regression layer. The regression layer casts the output size of the Resnet backbone, which is 1024, into 25 outputs. This is the number of outputs of the eye-tracking task and can be seen as 25 different output metrics displayed on the Tensorboard. Some examples of these outputs can be listed as overall loss, l2 regularization error, pupil and glint position error within average, p20, p50, or p80 percentile, etc.

The model is completed after adding the last layer of denormalization which denormalizes the 25 outputs in a range of [-1,1] into the already set up mean and standard deviation of pupil coordinates of the eye.

After taking the input, we need to preprocess and form the baseline model. This is done by first applying padding standardization and blurring the 200 by 200 images to be compatible with the ResNet model which has the input size of 224 by 224. Then this grayscale image is copied into 3 RGB channels. Secondly, we only take the feature maps of the backbone model without using its classifiers and apply the final block of Dropout with 0.2 probability and average pooling to prevent overfitting. In the last steps, the backbone gives 2048 outputs which should be cast into 25 different loss functions containing pupil and glint losses like percentiles, etc. Finally, we apply denormalization to the normalized values to get a higher range of values. By the end, we use this vector of values as input and get 25 denormalized outputs.

3.4 Network architectures

As the baseline backbone model chosen for this project is Resnet with 50 layers, the baseline model initially has a size of 100 MBs. Since only the feature network of the Resnet model is being used the model size is decreased to 94.6 MBs. The feature network refers to the intermediate layers in Resnet50 excluding the last fully connected layer. The goal is to obtain a model with a size less

than 1 Mb, while not losing too much accuracy as well as having an overall loss close to 1. Hence, other possibly smaller architectures are explored. Another model that also only makes use of the feature layer is the Mobilenetv2 thereby decreasing the total model size from 14.3 Mbs to 9.3 Mbs. In addition to these two models, Squeezenet and Shufflenet as depicted in Table A.1 are explored. Another modern and small-sized model that also obtains good accuracy is EfficientNet. EfficientNet is not implemented in PyTorch and as mentioned in the previous chapter EfficientNet has a lot of different versions with different layers. For this reason, the aforementioned networks are the ones to be explored using the various model compression methods. Since SqueezeNet makes a better candidate due to its size, EfficientNet and ShuffleNet networks are excluded from this project.

The depicted table shows different modern network architectures and their top 1 and 5 accuracy in the Imagenet dataset provided by Google. In top 1 accuracy, the predicted value should directly match the actual class where the top 5 accuracy, if one of the most 5 probable predicted classes matches the ground truth, is counted as accurate.

In the table, Resnet architectures have a fairly good accuracy but suffer from the model complexity. The Mobilenetv2 has an accuracy that is close to the accuracy of the Resnet architectures. The size of Mobilenetv2 also amounts to 10 percent of Resnet's model size. Since the aim is to explore smaller models MobileNetv2 and the Resnet baseline model with 50 layers will be explored. In addition, Shufflenet and SqueezeNet architectures can be decreased to a size of 5 MB although they lost some accuracy. When only using the feature maps of SqueezeNet it obtains a model size of 3MB, and since it has similar accuracy to ShuffleNet it will be explored as well.

3.5 Pruning

3.5.1 Unstructured Pruning

As the weights of any deep learning model can be visualized, it is apparent that lots of the weights are very close to zero. In this case, it is not essential to start with the baseline network and attempt to prune its weights since the model size is considered large and the model has lots of parameters which makes it inefficient. PyTorch´s pruning framework implements unstructured pruning as it zeroes the unwanted weights by applying global l1 and global ln pruning. In addition to this, a threshold pruning functionality is written as many of the weights are close to zero to see the effects of removing them.

3.5.2 Structured Pruning

After removing the weights which are close to zero the model size remains the same. The weights are attempted to be removed by setting them to zero, but this procedure has no effect since the model still stores the tensor that holds 32 bits of zeros after the floating point. Therefore, there is a need to remove these weights to reduce the model size and thereby decrease the inference time as well. Unlike the unstructured pruning methods implemented in PyTorch, there has been an attempt to remove the last layers and train the network without them to decrease the model size.

3.6 Distillation

In contrast to the previously described model distillation theory, there will not be a softmax layer applied at the end. The reason for this is that the prevalent task is an image regression task and the output in this case is of scalar values and not classes. There have been various suggestions made on what the appropriate loss function is for image regression tasks [19] [20]. In this project, the total loss function is calculated using a linear combination of distillation loss multiplied by the distillation factor and the actual loss. In Tobii´s DL framework, there is a function that calculates the overall loss for eye images, by combining glint and pupil loss. This function is used to obtain the actual loss. The distillation loss is measured using the mean-squared error (mse) loss between the outputs produced by the student model and the teacher model. The output of the model is optimized according to the total loss. Normally, the distillation technique is implemented before the softmax layer i.e. after the output of the backbone model. In the case of a classification task, the numerical values of the output produced by the backbone model would be normalized after the softmax layer. This would be done to obtain the class probabilities which would sum up to 1. The reason for this is that the distillation is more effective when the output is in non-normalized numerical values i.e. before the softmax layer. However, since the focus of this project is on eye-tracking which is a regression task, the softmax layer is not applied.

Algorithm 2 Distillation logic

```
1: function TRAIN_ONE_EPOCH(all_batches, kd_factor)
2:     epoch_loss ← 0
3:     function TRAIN_ONE_BATCH(one_batch, kd_factor)
4:         batch_losses ← []
5:         repeat
6:             real_loss ← overall_loss(ground_truth, student_prediction)
7:             distillation_loss ← mse_loss(teacher_prediction, student_prediction)
8:             batch_loss ← kd_factor * distillation_loss + (1 −
   kd_factor) * real_loss
9:             optimize(batch_loss)
10:            batch_losses ← append(batch_loss)
11:        until all batches are completed.
12:        return batch_losses
13:    epoch_loss ← avg(batch_losses)
14:    return epoch_loss
```

3.7 Quantization

Since the Pytorch framework only allows 8-bit quantization to be applied using a 32-bit floating point, this is the conversion of choice in this project. Implementations of bitwise operations which could be used to save weights as tensors of integers are discarded due to time constraints. Another possibility would be to apply 2-16 bits quantization from 32 bits using the Tensorflow and Keras framework from PyTorch. This option is also discarded due to time constraints.

To start with, the data needs to be prepared for the quantized model in Pytorchs quantization modules. The model should have two additional layers, both pre-and post-quantization layers. To prepare the input, a quant layer is introduced. The quant layer converts data from 32 bits into 8 bits. A dequant layer is used to convert the 8-bit output of the quantized model into a 32-bit actual output.

For the quantization of the model, each layer and module are fused and propagated to the next step which prepares and converts the model into a quantized model. Lastly, the quantized model gets the 8-bit output and dequantizes it to the final 32-bit output. The final output forms the loss using the ground truth. The generated loss is compared with the loss of the non-quantized model to see if the gain of accuracy can also be achieved as well as the gain of the model size.

There exist two types of quantizations that are related to the prevalent research: post-static quantization and quantization-aware training. Both use the same fusing and conversion techniques to switch 32-bit floating models to 8-bit models to get the model size decreased. However post-static quantization can only be done after the training whilst quantization-aware training uses the training of the quantized model to adjust its output better. The drawback of the latter method is that Pytorch only allows the CPU to be used in quantization. Thus it takes 2 times the time to train the quantized model.

The mapping function for weight bits quantization is formulated below:

$$Q(x, scale, zero\_point) = round(\frac{x}{scale} + zero\_point) \tag{3.1}$$

which is taken from Pytorch documentation [21].

3.8 Evaluation metrics

In addition to the implementation of model compression techniques and training of different models, the evaluation script compares each model in terms of the size and complexity of each network. The size and complexity of the CNNs are evaluated by recording the metrics presented below:

1. The accuracy of the model:
    - Overall loss: loss function of the combination of the eye features i.e. pupil and glints as pixel difference
    - Pupil position error as pixel difference
    - Training and inference loss of each aforementioned loss function.
2. The size and complexity of the model:
    - The model size (in MB)
    - The number of parameters in the weights of the model
    - The time required for a single forward pass (inference or execution time in seconds)
    - The number of Floating Point Operations (G-Flops)
    - Sparsity ratio of the weights file

3.9 Experiments framework logic

By the end of the implementation, the final step is to run the experiment framework script which includes pre and post-model compression techniques. As depicted in the figure below, the logic of the experiment's framework can be seen thoroughly.

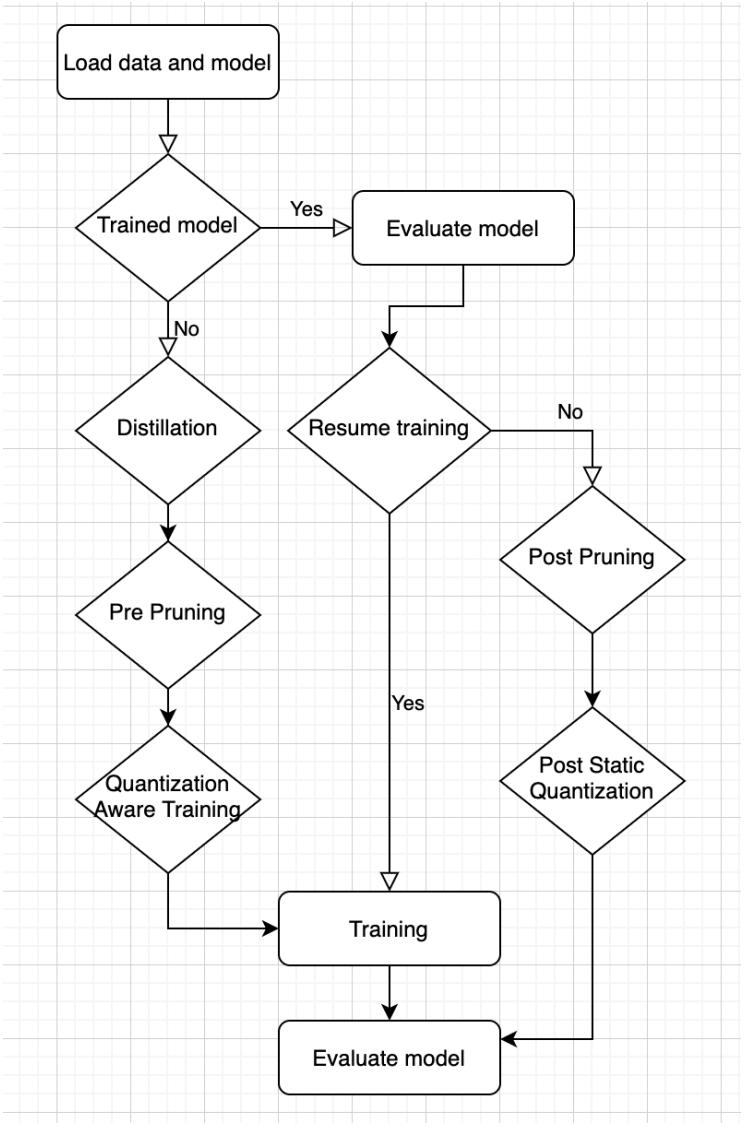


Figure 3.4: Experiments framework

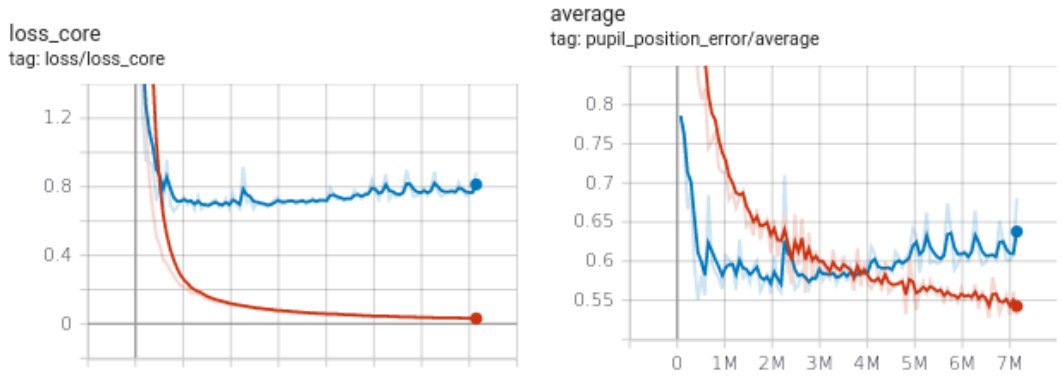
Chapter 4

Experiments and Results

The experiments are conducted using the same hyperparameters since the aim of the thesis is not concerned with hyperparameter tuning or optimization of the models. The hyperparameters used in all of the experiments are the following: 100 epochs, 0.2 Dropout probability, initial learning rate of 0.1, batch size of 16, and the size of the dataset is 6.6 GB. However, the weights file of the trained model which gives the best accuracy is saved to achieve the best results of that model in 100 epochs. Each experiment takes 60 minutes for each epoch when run on the development computer with a GeForce GTX 1080 GPU. Using a terminal computer equipped with 2 Quadro RTX 5000 GPUs decreased the training and validation time for each epoch by 20 percent. In that way, it takes 48 minutes for each epoch instead. This resulted in decreasing the total running time of each experiment from 4 days to 3 days. Since there exist approximately 73k and 59k images in the training and evaluation datasets for eye and nose recordings, training takes 4556 iterations and evaluation takes 3648 iterations for each considering that the selected batch size is 16.

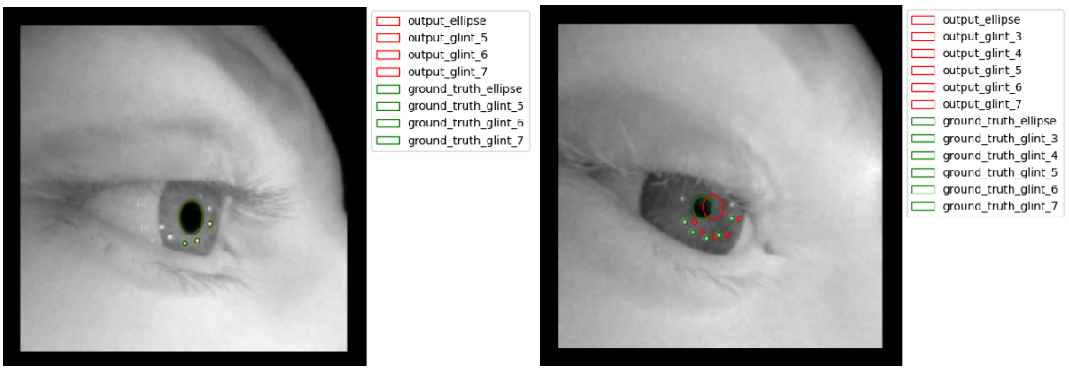
4.1 Baseline network

The ResNet50 network provided within Tobii´s DL framework is chosen as the baseline model for the conducted experiments. The following results show overall loss and pixel position error plots as well as the best and worst predictions of eye images in the validation dataset. In the plots, the red line shows the performance of the model on the training dataset whilst the blue line corresponds to those on the validation dataset. The first two figures represent the results for the eye images recorded from ear cameras whilst the last two figures include those recorded from nose cameras.



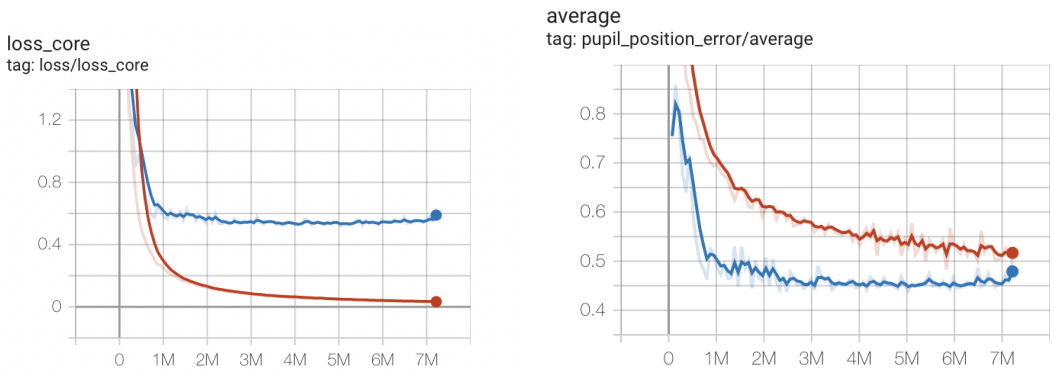
(a) Overall loss **(b)** Pixel position error

Figure 4.1: Overall loss and pixel position error plots vs. iteration count for eye data recorded from ear cameras



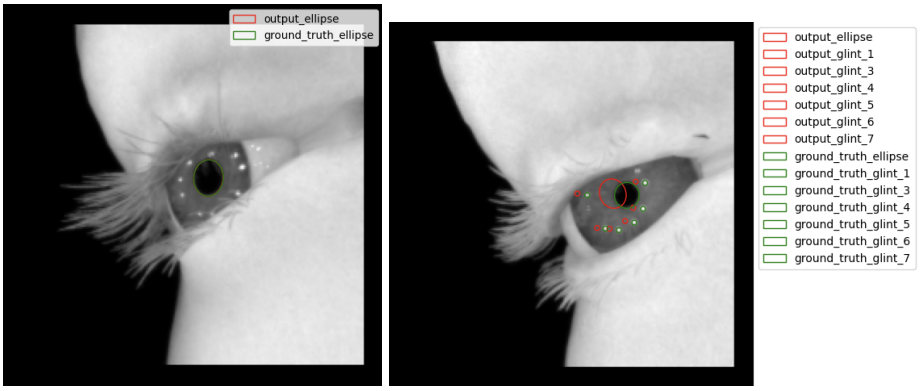
(a) Best prediction **(b)** Worst prediction

Figure 4.2: Best and worst predictions for eye images from ear cameras in validation dataset



(a) Overall loss **(b)** Pixel position error

Figure 4.3: Overall loss and pixel position error plots vs. iteration count for nose data recorded from nose cameras



(a) Best prediction **(b)** Worst prediction

Figure 4.4: Best and worst predictions for eye images from nose cameras in validation dataset

As shown in the worst predictions, the error does not exceed 1 pixel. Hence, the validation loss resulted in being closer to and less than 1.

Models	Pretrained	Training loss	Validation loss
Ear Resnet-50	Yes	0.06	0.77
Ear Resnet-50	No	0.06	0.99
Nose Resnet-50	Yes	0.06	0.55
Nose Resnet-50	No	0.19	1.14

Table 4.1: Overall loss of baseline model for training and validation datasets

Since better validation loss results are obtained when the pre-trained weights for the ImageNet dataset are not set, all consequent experiments are conducted with the randomly initialized weights at the beginning. After conducting these experiments, the remaining experiments are done with pre-trained weights which are enabled by setting the pretrained argument of the available networks in PyTorch to True. The results are better when the pre-trained weights for the Imagenet dataset are used within 100 epochs. Therefore, the baseline models are set as Resnet 50 with pre-trained weights which resulted in 0.77 and 0.55 for the ear and nose models respectively.

4.2 Various network architectures

Motivation The motivation of the experiment is to compare the accuracy of different network architectures and shift the baseline model to a smaller model with an acceptable accuracy even though the loss is increased.

Result The experiments generated good accuracy for the Resnet and MobileNet models. A smaller-sized model is obtained from SqueezeNet measuring 3MB without losing too much accuracy. The loss increased from 0.7 to 0.96 in-ear images, and from 0.55 to 1.57 in the nose images. Therefore the baseline model is replaced by Squeezenet as it is smaller and there is an attempt to achieve an even smaller sized model using model compression methods.

As the Resnet 50 of 100 MB is believed to have excessive weights for the eye-tracking task, it is necessary to try out different smaller architectures which are considered to give better results in terms of memory and time complexity.

Models	Training loss	Validation loss	Model Size (MB)	Number of parameters (Millions)
Resnet-50	0.06	0.77	94	23.55
Resnet-18	0.12	0.8	44	11.19
MobileNetv2	0.5	0.72	9.3	2.26
SqueezeNet	0.43	0.96	3	0.73

Table 4.2: Overall loss of different network architectures for ear dataset

Models	Training loss	Validation loss	Model Size (MB)	Number of parameters (Millions)
Resnet-50	0.06	0.55	94	23.55
Resnet-18	0.09	0.72	44	11.19
MobileNetv2	0.24	0.71	9.3	2.26
SqueezeNet	0.68	1.57	3	0.73

Table 4.3: Overall loss of different network architectures for nose dataset

Since a fairly good accuracy is obtained with SqueezeNet even though it increases in comparison to bigger architectures like Resnet. The gain in terms of model size and complexity is considered good when compared to the small accuracy loss.

4.3 Pruning

Motivation The baseline models before pruning are kept as Squeezenet since it generated a loss closer to 1 for both the ear- and the nose datasets. This experiment aims to decrease the model size further from 3 MB to a model of a size less than 1 MB.

4.3.1 Unstructured pruning

a. Global pruning

Motivation The pruning is started by experimenting with Pytorch's existing method for global pruning with the L1 norm to see how the sparsity ratio would affect the inference loss for the Squeezenet to be able to decrease the model size.

Results The results showed that when using unstructured pruning of the L1 norm, it is not possible to decrease the model size, as the pruning in Pytorch only zeroed the weights and stored them in 32-bit floating point numbers. Instead, pruning can achieve sparser models and if a way to remove the weights without losing the structure of the model was found, it could also have reduced the model size.

Amount	Sparsity Ratio	Inference loss
0.1	10	0.91
0.2	20	5.2
0.3	30	21.23
0.4	40	31.1

Table 4.4: Results of unstructured global pruning for ear dataset

Amount	Sparsity Ratio	Inference loss
0.1	10	1.55
0.2	20	6.3
0.3	30	23.1
0.4	40	41.1

Table 4.5: Results of unstructured global pruning for nose dataset

b. Threshold pruning

Motivation Some weights of the network are close to 0. Therefore, the rationale behind this experiment is to examine the effect of removing or pruning the weights that have smaller values with the specified threshold values as input, to zero a high percentage of the weights.

Threshold	Sparsity Ratio	Inference loss
0.1	40	12
0.01	30	9.5
0.001	15	5.57
0.0001	10	0.94

Table 4.6: Threshold pruning results for ear dataset

Threshold	Sparsity Ratio	Inference loss
0.1	40	32
0.05	30	18
0.01	20	6.8
0.001	10	1.51

Table 4.7: Threshold pruning results for nose dataset

Results The initial idea is to freeze all the weights assigned to zero after the unstructured pruning and then apply retraining to increase their accuracy. However, as these methods of unstructured pruning do not reduce either the model size or the inference time, retraining or attempting distillation on these networks is skipped to save time for the remaining experiments.

4.3.2 Structured pruning

a. Channel Ln pruning

Motivation The motivation is to decrease the model size by removing or zeroing the whole channel, thus the model size would be removed.

Results Zeroing the whole channel weight values does not decrease the model size as zeroes are also represented in 32 bits and are stored when saving the model. As a result, pruning may improve the performance if the sparsity ratio is set to a reasonable amount like 10 percent. But the loss would increase too much if more weights were removed. Also, structured pruning showed better results than unstructured pruning and in practice one can end up with reduced model sizes if you delete these zeroed weights.

Amount	Sparsity Ratio	Inference loss
0.1	10	0.909
0.2	20	4.3
0.3	30	14.2
0.4	40	25.3

Table 4.8: The impact of structured pruning via Ln pruning method for ear dataset

Amount	Sparsity Ratio	Inference loss
0.1	10	1.5
0.2	20	5.2
0.3	30	15.32
0.4	40	27.45

Table 4.9: The impact of structured pruning via Ln pruning method for nose dataset

b. Layer pruning

Motivation In contrast to the previous experiments which are unable to reduce the model size, the idea behind this experiment is to attempt to train the model in which the final layers are removed. The model until column that is shown in the tables refers to the models that have the last conv layer specified in the rows. For example, the model until layer 11 refers to the model in which layers after 11 are removed to decrease the model size and see the effects of the latter layers in training.

Results As a result, there is a high decrease in model size from 3 MB to as low as 0.1 MB. But this model suffers from an increase in the loss which goes up to 26. Instead, a smaller model with the layer 9 pruned version of 1 mb model size and a reasonable loss of 1.46 is used to proceed.

Models until	Model Size (MB)	Number of parameters	Inference time in CPU (msec)	Inference loss
Layer 11	2.2	0.54	116.35	1.32
Layer 10	1.4	0.35	109.02	1.4
Layer 9	1.0	0.24	105.28	1.46
Layer 7	0.5	0.13	94.3	5.68
Layer 6	0.3	0.08	89.06	8
Layer 4	0.1	0.03	74.34	26

Table 4.10: Layer pruning effect on SqueezeNet with no pre-trained weights for ear dataset

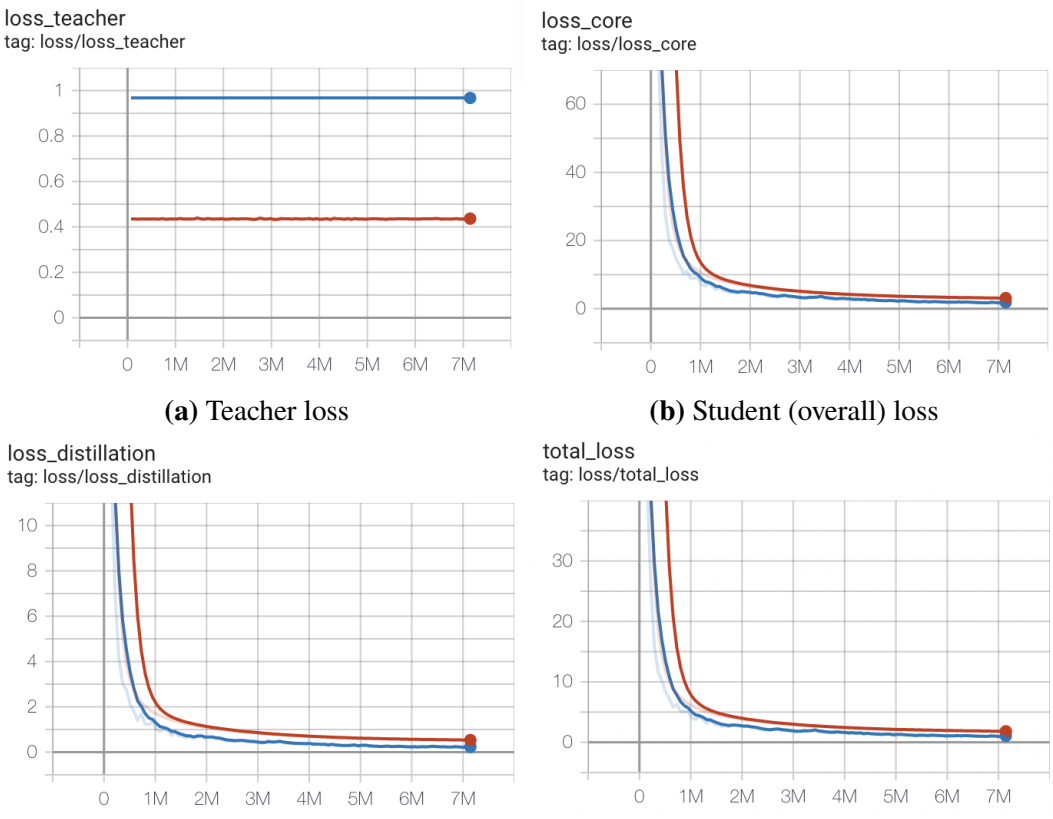
Models until	Pretrained	Model Size (MB)	Number of parameters	Inference time in CPU (msec)	Inference loss
Layer 11	Yes	2.2	0.54	60.16	2.57
Layer 10	Yes	1.4	0.35	75.54	3.16
Layer 9	Yes	1.0	0.24	64.96	3.29
Layer 11	No	2.2	0.54	56.16	2.29
Layer 10	No	1.4	0.35	107.82	2.11
Layer 9	No	1.0	0.24	73.79	5.02

Table 4.11: Layer pruning effect on SqueezeNet for nose dataset

4.4 Distillation

Motivation The baseline network is chosen as the teacher model since it has the best accuracy for its best scoring model. The student models are all models for which the experiments above are done without distillation. Therefore, it is of value to conduct experiments with these network architectures with the distillation model compression method. The motivation behind this experiment is to see whether or how much the model loss improved by using the distillation method compared to the default training.

Results Overall, the results are not as expected as they do not decrease the loss in most of the models. It only decreased the loss of Squeezenet layer 9 in the nose datasets from section 3.3 to nearly a loss of 3.



(a) Teacher loss

(b) Student (overall) loss

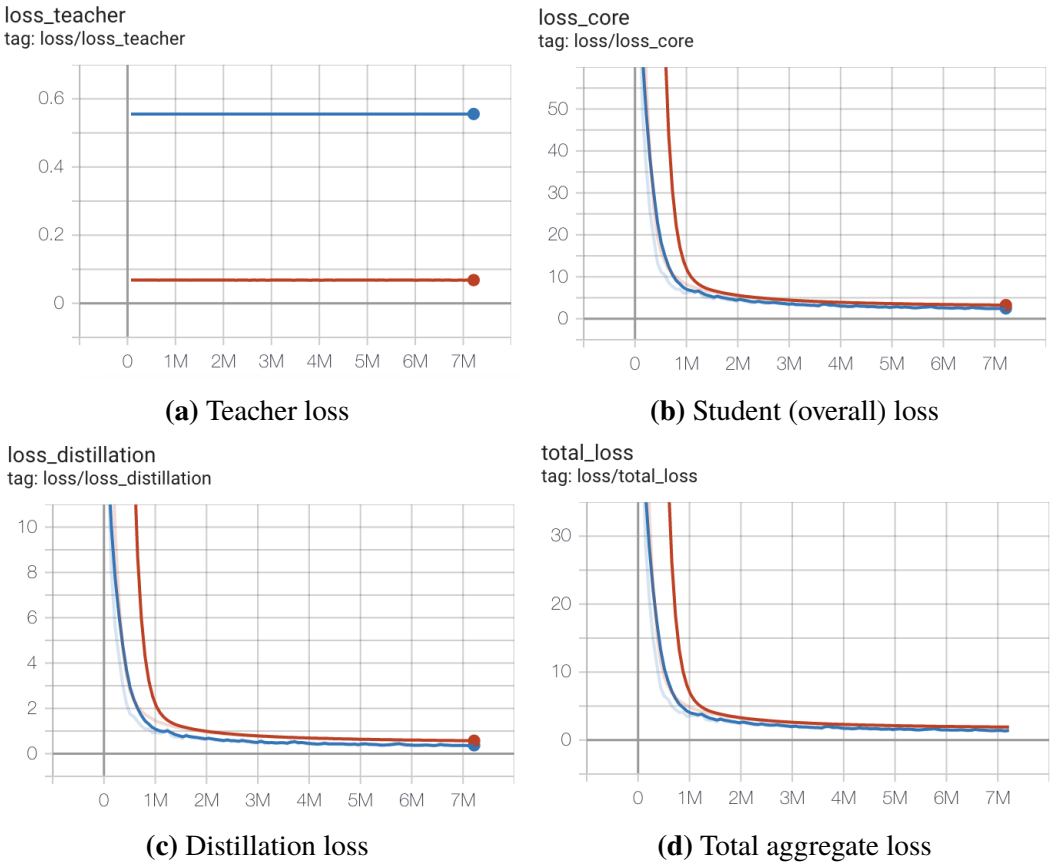
(c) Distillation loss

(d) Total aggregate loss

Figure 4.5: Loss plots for eye images from ear cameras in validation dataset when teacher network is SqueezeNet (3 MB) and student network is SqueezeNet with 9 layers (0.955 MB)

Teacher Network	Student Network	Distillation factor	Training overall loss	Validation overall loss
ResNet-50	MobileNetv2	0.5	0.22	0.73
MobileNetv2	SqueezeNet	0.5	0.37	1.27
MobileNetv2	SqueezeNet	0.8	0.448	1.4
MobileNetv2	SqueezeNet	0.95	0.444	1.51
MobileNetv2	SqueezeNetLayer9	0.5	1.08	2.38
SqueezeNet	SqueezeNetLayer9	0.5	1.12	2.06

Table 4.12: Evaluation of different student models for training and validation of ear datasets



(a) Teacher loss

(b) Student (overall) loss

(c) Distillation loss

(d) Total aggregate loss

Figure 4.6: Loss plots for eye images from nose cameras in validation dataset when teacher network is ResNet-50 and student network is SqueezeNet with 10 layers (1.4 MB)

Teacher Network	Student Network	Distillation factor	Training overall loss	Validation overall loss
ResNet-50	SqueezeNetLayer10	0.5	0.89	2.46
ResNet-50	SqueezeNetLayer9	0.5	1.11	3.02
MobileNetv2	SqueezeNetLayer9	0.5	1.22	3.08
SqueezeNet	SqueezeNetLayer9	0.5	1.44	3.05

Table 4.13: Evaluation of different student models for training and validation of nose datasets

4.5 Quantization

Motivation Via post-static quantization, quantizing the weights of the model after training, the motivation is to decrease the model size further without decreasing the accuracy too much.

Model	Model size (MB)	Inference time (CPU)	Inference loss
ResNet-50	23	923	5
ResNet-18	11	798	10
MobileNetv2	2.5	358.2	22
SqueezeNet	0.8	270	33

Table 4.14: Post-static-quantization effect on the ear models

Model	Model size (MB)	Inference time (CPU)	Inference loss
ResNet-50	23	923	7
ResNet-18	11	798	15
MobileNetv2	2.5	358.2	28
SqueezeNet	0.8	270	43

Table 4.15: Post-static-quantization effect on the nose models

Results Post-static quantization in Pytorch does not decrease the accuracy too much as the predicted output class is decided by the highest probability in the image classification tasks (refer to post). In these tasks, the quantization nearly led to a 0.01 percent accuracy decrease.

In contrast to the image classification tasks, the quantization might not perform the same since this is an image regression task. Rather than taking the highest probable class, any single deviation in the output is important for regression losses. It resulted in a huge information loss from 0.96 to 33 for example in the image regression task. However, a strong model compression is achieved as the model decreases from 3 MB to 0.8 MB.

4.6 Analysis

The overall plots of the ear and the nose models are depicted in this section. The DL models inside these two boxes can be thought of as better performing and smaller models with their inference time lower than 500 and overall loss closer to or lower than 1-pixel pupil as well as glint difference error for both the ear and the nose images.

Ear dataset

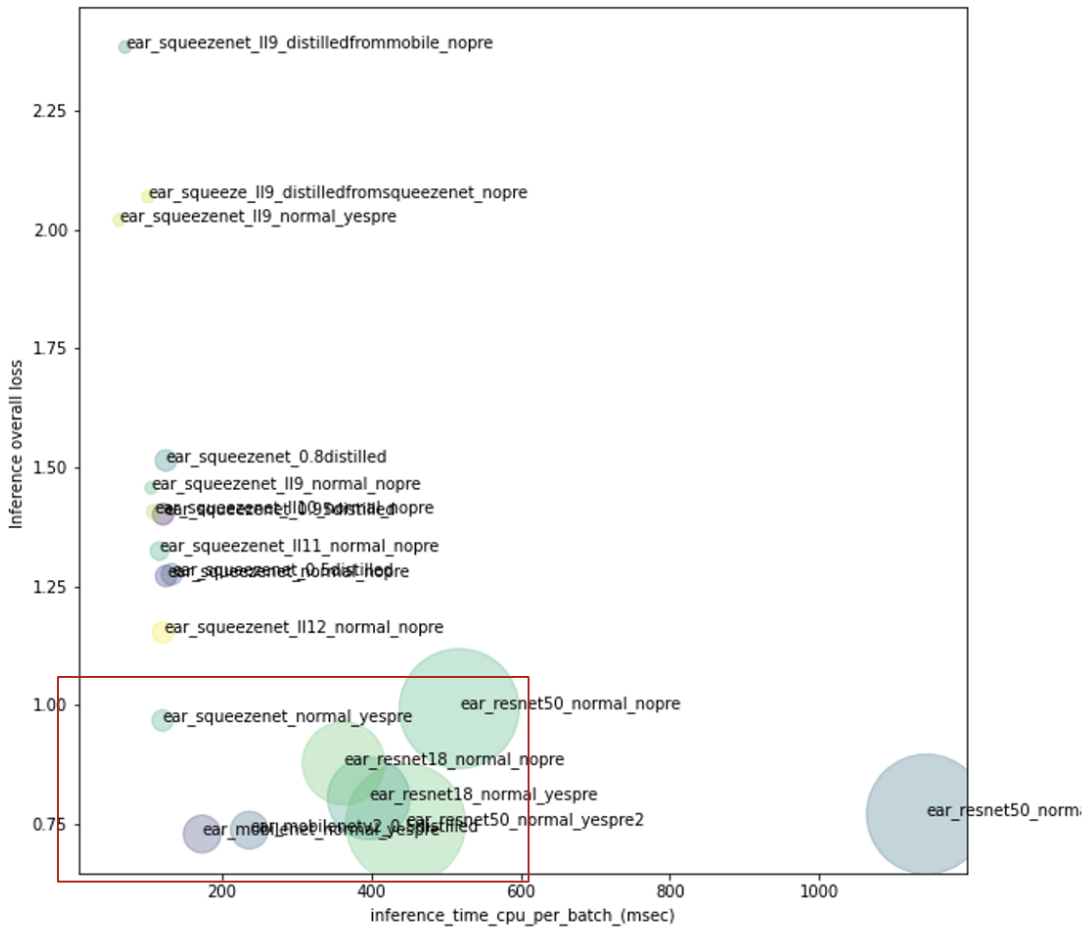


Figure 4.7: Inference time on CPU vs. overall loss of the ear models

Nose dataset

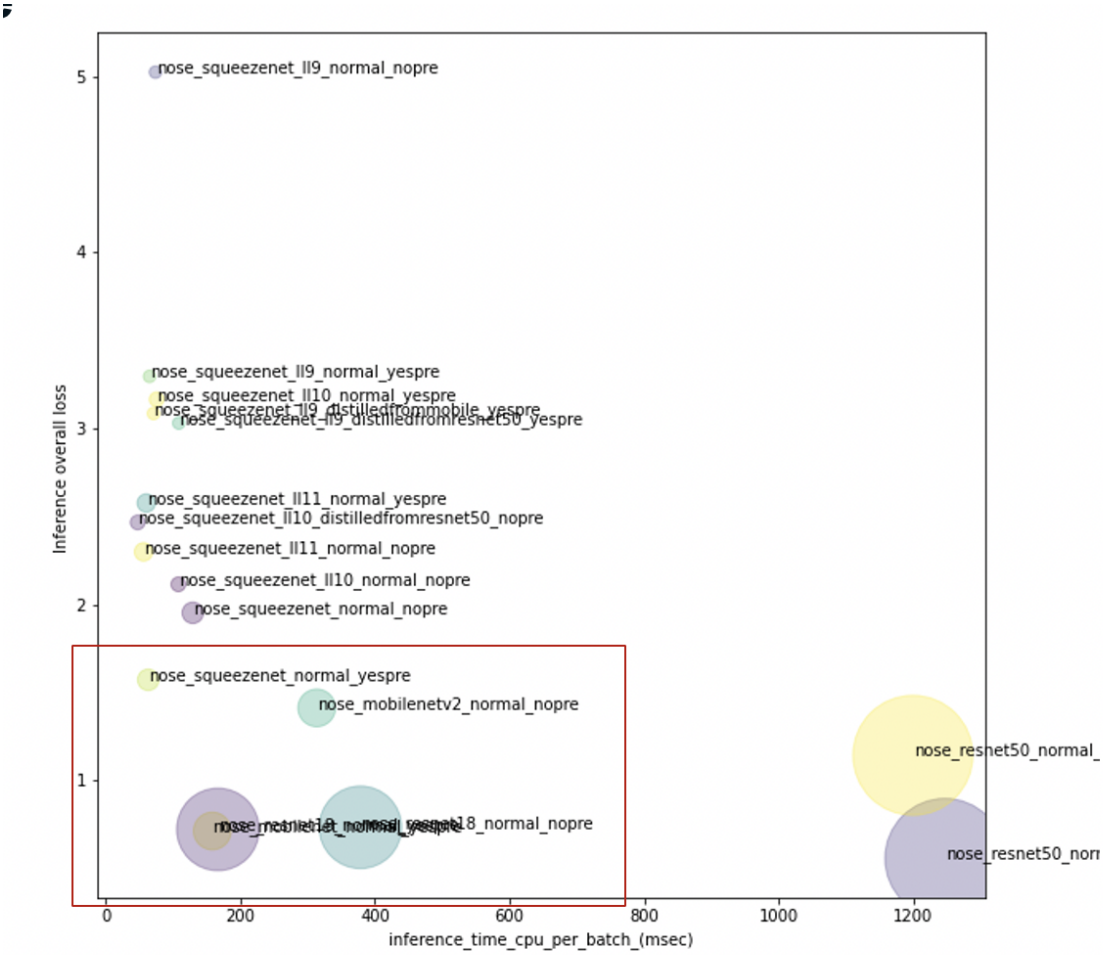


Figure 4.8: Inference time on CPU vs. overall loss of the nose models

Chapter 5

Discussion

The bigger dataset the model is trained with the lower loss it has. If the dataset is increased from 1GB to 7GB, the pixel difference can be lowered as much as 1.5 even with the smaller network architectures whose model size is below 1 MB with layer pruning. The bigger the dataset the model is trained with the lower the loss it has. However, the sample dataset of 70 MB will have a similar loss to the largest validation dataset. So the dataset is fairly homogeneous.

The small and thin architectures are hard to train. When bigger models are attempted to be compressed into smaller models, they become hard to train as the overall loss for the evaluation dataset remains lower than the training loss even after 100 epochs. This concludes that the model should be trained more to converge whilst the bigger models like ResNet-50 are already converged as its training loss became lower than the validation loss.

Pruning and quantization methods are straightforward techniques to decrease the model size. These two techniques are effective in forming thin and small architectures that have less information as the number of bits after the floating point is kept as 8 bits rather than 32 bits. Furthermore, bigger networks have excessive weight information as the overall loss of these smaller networks does not exceed much from the baseline network.

Distillation can be used to converge the small models faster. If the model includes a thin and small network architecture and is hard to converge by default training, distillation is a good method to train and converge the model

faster. It requires less number of epochs to converge the student model if provided by a larger teacher network than by the default training.

Distillation is not a good method to boost the accuracy when the model can converge. When the model can converge by the default training, the distillation method neither improves the accuracy of the prediction nor decreases the model size. So the default training is a better approach if the need is not to converge the model faster.

Quantization suffers in inference loss in image regression tasks. Post-static quantization is a promising method for decreasing the model size, thus inference time and CPU memory usage, and even it improves or does not decrease the accuracy for image classification tasks. However, it is not applicable for image regression tasks since it increases the overall loss significantly.

Chapter 6

Conclusions

In this paper, the research is conducted to compress the Deep Learning models which solve an image regression task called eye tracking, which is defined as prediction of localization of pupil and glints of the eye images recorded from the Tobii eye tracking glass product. The compression of the DL models is needed due to the memory and CPU limit of the Tobii eye-tracking glass which is an embedded product. Therefore, model compression techniques like pruning, distillation, and quantization are researched and conducted to achieve this goal without losing too much accuracy even improving loss if possible.

The dataset includes synthetic eye images recorded from 4 cameras placed at the left and right sides of the ear and nose. The dataset includes training and validation datasets with 73k training and 59k evaluation images. The challenge is the image regression task which tries to localize the pupil and glints ellipses of the eye in the synthetic images. Thus, the overall loss function algorithm is provided by Tobii which involves the combination of pixel loss of the pupil and glints ellipses of the eye between the ground truth and the prediction of the ellipses pixels.

The experiments start with the baseline model of Resnet50 94 MB, with a dataset of 73k training and 59k evaluation images which scored 0.77 as overall loss calculated with pupil and glints. Then, the experiment continued with the various models: Resnet18, Mobilenetv2, and Squeezenet to modify the baseline into a smaller model in case the information loss of the overall loss can be tolerable in consideration of the gain of the model size. Later, the model compression techniques are performed to achieve the goal of at most 1 MB of small models without losing too much accuracy. Unstructured pruning methods like global and threshold pruning improve the overall loss a bit but it does not decrease the model size since they only zero the weights, while structured

pruning methods like channel and especially layer pruning decrease the model size by losing the accuracy from around 1 to 1.44 for SqueezeNet model. For the distillation compression methods, knowledge-based distillation with a distillation factor of 0.5 with many combinations of teacher and student networks chosen from the experimented network architecture stated above is performed. It neither improves the loss nor the model size of the student model, however, it could be used to converge the model faster without awaiting more epochs with the default training. In addition, post-static quantization is performed to decrease the model size further from 3 MB to less, but it increases the overall loss significantly even though the model size is decreased to 0.8 MB. So, this method is not a promising method for image regression tasks since it decreases the accuracy too much although it works well in image classification tasks.

Consequently, this paper achieved the compression of the models listed below into smaller ones. The compressed models are described with the following: model name, inference time, memory usage, and loss. For the eye images recorded from ear cameras, the baseline model is ResNet-50 whilst the last optimal model is LayerPruned SqueezeNet with 9 layers. In the end, the model size is decreased from 94 to 1 MB, inference time per batch from 446.3 to 105 msec, CPU memory usage from 1340 to 211 MB, and overall loss from 0.77 to 1.44 in the validation dataset. For the eye images recorded from nose cameras, the baseline model is ResNet-50 whilst the last optimal model is LayerPruned SqueezeNet with 10 layers. In the end, the model size is decreased from 94 to 1.7 MB, inference time per batch from 446.3 to 105 msec, CPU memory usage from 1340 to 211 MB, and overall loss from 0.55 to 1.7 in the validation dataset.

Chapter 7

Future work

For future work, the below-mentioned points and topics are suggested to be investigated further.

Deep-compressed Squeezenet is a promising network to use. In addition to Squeezenet's good performance and low accuracy loss compared to AlexNet of 250 MB, Han et. al proposes a technique called DeepCompression [18] which can reduce SqueezeNet up to 0.4 MB while preserving its accuracy. In addition, a layer-pruned version of this model can be used to achieve deep learning models of size lower than 100 KB. DeepCompression on Squeezenet can achieve a 0.4 MB sized model from the 3 MB model by using pruning Huffman encoding and 6-bit quantization. This particular code is written in Caffe, hence, it is discarded from this project.

Image preprocessing techniques like downsampling can be used to boost the accuracy. This paper is not concerned with data preprocessing techniques but the overall loss can be significantly improved by image preprocessing methods like downsampling where the image pixels inside a kernel (n x n) are equalized with the pixel value in the center, where n can be chosen as 3.

Normalized Pixel Difference can be used to boost the accuracy further as an advanced image preprocessing technique. Traditional ML methods like Normalized pixel differences (NPD) work well with image regression tasks like eye-tracking so an imitation of CNN can be designed to mimic the behavior of these traditional ML methods so that the model size is kept as small and thus can be deployed to embedded devices with ease.

SparseConvNet can be used to decrease the number of weights. Changing the deep convolution algorithm calculated between pixels and the kernel can be changed to a lightweight CPU algorithm called SparseConvNets where only the specific and valuable convolutions can be taken between image pixels and the kernel. Therefore, the inference time and model size can be decreased since the model includes less number of weights.

Performing NAS or AutoML to find the smallest model with better accuracy. NAS can be performed to find the optimal network architecture with the desirable model size by not decreasing the accuracy of the predictions. Even AutoML can be used to select the best ML pipeline so that it would find the best model with the optimal preprocessing techniques and hyperparameters. These two techniques can automate the experiment processes without needing to investigate each ML step.

References

[1] "Tobii is the world leader in eye tracking," Oct. 2015. [Online]. Available: https://www.tobii.com/group/about/

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016. doi: 10.1109/CVPR.2016.90. ISBN 978-1-4673-8851-1 pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," *arXiv:1801.04381 [cs]*, Mar. 2019, arXiv: 1801.04381. [Online]. Available: http://arxiv.org/abs/1801.04381

[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *arXiv:1808.05377 [cs, stat]*, Apr. 2019, arXiv: 1808.05377. [Online]. Available: http://arxiv.org/abs/1808.05377

[5] M. Wistuba, A. Rawat, and T. Pedapati, "A Survey on Neural Architecture Search," *arXiv:1905.01392 [cs, stat]*, Jun. 2019, arXiv: 1905.01392. [Online]. Available: http://arxiv.org/abs/1905.01392

[6] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," p. 106622, Jan. 2019. [Online]. Available: https://doi.org/10.1016%2Fj.knosys.2020.106622

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv:2010.11929 [cs]*, Oct. 2020, arXiv: 2010.11929. [Online]. Available: http://arxiv.org/abs/2010.11929

[8] "Tobii dynavox assistive technology for communication," Sep. 2015. [Online]. Available: https://www.tobiidynavox.com/pages/about-us

[9] D. Hebb, "The organization of behavior; a neuropsycholocigal theory," pp. 62–78.

[10] "What is eye tracking?" Oct. 2015. [Online]. Available: https://tech.tobii.com/technology/what-is-eye-tracking/

[11] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015.

[12] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network- a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679–688, 2018. doi: https://doi.org/10.1016/j.procs.2018.05.069 International Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918308019

[13] S.-H. Tsang, "Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection)," Mar. 2019. [Online]. Available: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv:1704.04861 [cs]*, Apr. 2017, arXiv: 1704.04861. [Online]. Available: http://arxiv.org/abs/1704.04861

[15] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design," *arXiv:1807.11164 [cs]*, Jul. 2018, arXiv: 1807.11164. [Online]. Available: http://arxiv.org/abs/1807.11164

[16] A. Geifman, "Efficient Inference in Deep Learning — Where is the Problem?" Jun. 2020. [Online]. Available: https://towardsdatascience.com/efficient-inference-in-deep-learning-where-is-the-problem-4ad59434fe36

[17] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *arXiv:1602.07360*

[cs], Nov. 2016, arXiv: 1602.07360. [Online]. Available: http://arxiv.org/abs/1602.07360

[18] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv:1510.00149 [cs]*, Feb. 2016, arXiv: 1510.00149. [Online]. Available: http://arxiv.org/abs/1510.00149

[19] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv:1503.02531 [cs, stat]*, Mar. 2015, arXiv: 1503.02531. [Online]. Available: http://arxiv.org/abs/1503.02531

[20] L. Wang and K.-J. Yoon, "Knowledge Distillation and Student-Teacher Learning for Visual Intelligence: A Review and New Outlooks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. doi: 10.1109/TPAMI.2021.3055564 ArXiv: 2004.05937. [Online]. Available: http://arxiv.org/abs/2004.05937

[21] "Quantization -pytorch." [Online]. Available: https://pytorch.org/docs/stable/quantization.html#quantized-model

[22] X. Cao, Y. Wei, F. Wen, and J. Sun, "Face Alignment by Explicit Shape Regression," p. 8.

[23] S. Tibshirani and H. Friedman, "Valerie and Patrick Hastie," p. 764.

[24] C. Matsoukas, "Model Distillation for Deep-Learning-Based Gaze Estimation," p. 99.

[25] W. Sung, "Investigating minimal Convolution Neural Networks (CNNs) for realtime embedded eye feature detection," p. 74.

[26] J. Wu, H. Ren, Y. Kong, C. Yang, L. Senhadji, and H. Shu, "COMPRESSING COMPLEX CONVOLUTIONAL NEURAL NETWORK BASED ON AN IMPROVED DEEP COMPRESSION ALGORITHM," p. 5.

[27] Baoyuan Liu, Min Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse Convolutional Neural Networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, Jun. 2015. doi: 10.1109/CVPR.2015.7298681. ISBN 978-1-4673-6964-0 pp. 806–814. [Online]. Available: http://ieeexplore.ieee.org/document/7298681/

[28] C. Choy, J. Gwak, and S. Savarese, "4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks," *arXiv:1904.08755 [cs]*, Jun. 2019, arXiv: 1904.08755. [Online]. Available: http://arxiv.org/abs/1904.08755

[29] N. Markuš, M. Frljak, I. S. Pandžić, J. Ahlberg, and R. Forchheimer, "Eye pupil localization with an ensemble of randomized trees," *Pattern Recognition*, vol. 47, no. 2, pp. 578–587, Feb. 2014. doi: 10.1016/j.patcog.2013.08.008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0031320313003294

[30] P. Dollar, P. Welinder, and P. Perona, "Cascaded pose regression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. San Francisco, CA, USA: IEEE, Jun. 2010. doi: 10.1109/CVPR.2010.5540094. ISBN 978-1-4244-6984-0 pp. 1078–1085. [Online]. Available: http://ieeexplore.ieee.org/document/5540094/

[31] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH: IEEE, Jun. 2014. doi: 10.1109/CVPR.2014.241. ISBN 978-1-4799-5118-5 pp. 1867–1874. [Online]. Available: https://ieeexplore.ieee.org/document/6909637

[32] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," p. 39, 1999.

[33] H. Shah, A. Khare, N. Shah, and K. Siddiqui, "KD-Lib: A PyTorch library for Knowledge Distillation, Pruning and Quantization," *arXiv:2011.14691 [cs]*, Nov. 2020, arXiv: 2011.14691. [Online]. Available: http://arxiv.org/abs/2011.14691

[34] M. Lofqvist and J. Cano, "Accelerating Deep Learning Applications in Space," *arXiv:2007.11089 [cs, eess]*, Jul. 2020, arXiv: 2007.11089. [Online]. Available: http://arxiv.org/abs/2007.11089

[35] U. Ozbulak, "utkuozbulak/pytorch-cnn-visualizations," May 2021, original-date: 2017-10-21T07:25:07Z. [Online]. Available: https://github.com/utkuozbulak/pytorch-cnn-visualizations

[36] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks

via Gradient-based Localization," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, Feb. 2020. doi: 10.1007/s11263-019-01228-7 ArXiv: 1610.02391. [Online]. Available: http://arxiv.org/abs/1610.02391

[37] H. Peterson, "An Overview of Model Compression Techniques for Deep Learning in Space," Sep. 2020. [Online]. Available: https://medium.com/gsi-technology/an-overview-of-model-compression-techniques-for-deep-learning-in-space-3fd8d4ce84e5

[38] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the State of Neural Network Pruning?" *arXiv:2003.03033 [cs, stat]*, Mar. 2020, arXiv: 2003.03033. [Online]. Available: http://arxiv.org/abs/2003.03033

[39] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," *arXiv:1506.02626 [cs]*, Oct. 2015, arXiv: 1506.02626. [Online]. Available: http://arxiv.org/abs/1506.02626

[40] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv:1710.01878 [cs, stat]*, Nov. 2017, arXiv: 1710.01878. [Online]. Available: http://arxiv.org/abs/1710.01878

[41] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to Prune Filters in Convolutional Neural Networks," *arXiv:1801.07365 [cs]*, Jan. 2018, arXiv: 1801.07365. [Online]. Available: http://arxiv.org/abs/1801.07365

[42] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, Feb. 2020. doi: 10.1007/s11263-019-01228-7 ArXiv: 1610.02391. [Online]. Available: http://arxiv.org/abs/1610.02391

[43] K. Chahal, "How to Quantise an MNIST network to 8 bits in Pytorch (No retraining required) from scratch." Oct. 2019. [Online]. Available: https://karanbirchahal.medium.com/how-to-quantise-an-mnist-network-to-8-bits-in-pytorch-no-retraining-required-from-scratch-39f6

[44] "Concept of Quantization - Tutorialspoint." [Online]. Available: https://www.tutorialspoint.com/dip/concept_of_quantization.htm

[45] "Quantization - Neural Network Distiller." [Online]. Available: https://intellabs.github.io/distiller/quantization.html

[46] M. Sahni, "Making Neural Nets Work With Low Precision," Jun. 2018. [Online]. Available: /post/quantization-in-tflite/

[47] "Knowledge Distillation - Neural Network Distiller." [Online]. Available: https://intellabs.github.io/distiller/knowledge_distillation.html

[48] "Rank (linear algebra)," Mar. 2021, page Version ID: 1012968371. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rank_(linear_algebra)&oldid=1012968371

[49] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up Convolutional Neural Networks with Low Rank Expansions," in *Proceedings of the British Machine Vision Conference 2014*. Nottingham: British Machine Vision Association, 2014. doi: 10.5244/C.28.88. ISBN 978-1-901725-52-0 pp. 88.1–88.13. [Online]. Available: http://www.bmva.org/bmvc/2014/papers/paper073/index.html

[50] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua, "Learning Separable Filters," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR, USA: IEEE, Jun. 2013. doi: 10.1109/CVPR.2013.355. ISBN 978-0-7695-4989-7 pp. 2754–2761. [Online]. Available: http://ieeexplore.ieee.org/document/6619199/

[51] "Google Colaboratory." [Online]. Available: https://colab.research.google.com/drive/1oDfcLRz2AIgsclkXJHj-5wMvbylr4Nxz#scrollTo=xXkTAJ9ws1Y6

[52] S.-H. Tsang, "Review: MobileNetV1 — Depthwise Separable Convolution (Light Weight Model)," Feb. 2021. [Online]. Available: https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

[53] S. Liao, A. K. Jain, and S. Z. Li, "A Fast and Accurate Unconstrained Face Detector," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 211–223, Feb. 2016. doi: 10.1109/TPAMI.2015.2448075 ArXiv: 1408.1656. [Online]. Available: http://arxiv.org/abs/1408.1656

[54] S.-H. Tsang, "Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification)," Oct. 2020. [Online]. Available: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e

[55] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014, arXiv: 1409.4842. [Online]. Available: http://arxiv.org/abs/1409.4842

[56] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv:1512.00567 [cs]*, Dec. 2015, arXiv: 1512.00567. [Online]. Available: http://arxiv.org/abs/1512.00567

[57] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv:1404.5997 [cs]*, Apr. 2014, arXiv: 1404.5997. [Online]. Available: http://arxiv.org/abs/1404.5997

[58] S.-H. Tsang, "Review: YOLOv1 — You Only Look Once (Object Detection)," Mar. 2019. [Online]. Available: https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89

[59] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, Jun. 2016. doi: 10.1109/CVPR.2016.91. ISBN 978-1-4673-8851-1 pp. 779–788. [Online]. Available: http://ieeexplore.ieee.org/document/7780460/

[60] S.-H. Tsang, "Review: AlexNet, CaffeNet — Winner of ILSVRC 2012 (Image Classification)," Dec. 2020. [Online]. Available: https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. doi: 10.1145/3065386. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386

[62] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in *2018*

IEEE/CVF Conference on Computer Vision and Pattern Recognition. Salt Lake City, UT: IEEE, Jun. 2018. doi: 10.1109/CVPR.2018.00716. ISBN 978-1-5386-6420-9 pp. 6848–6856. [Online]. Available: https://ieeexplore.ieee.org/document/8578814/

[63] Song, "songhan/SqueezeNet-Deep-Compression," Apr. 2021, original-date: 2016-04-06T18:29:54Z. [Online]. Available: https://github.com/songhan/SqueezeNet-Deep-Compression

[64] "mightydeveloper/Deep-Compression-PyTorch: PyTorch implementation of 'Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding' by Son..." [Online]. Available: https://libs.garden/python/mightydeveloper/Deep-Compression-PyTorch

[65] mj, "bemova/Deep-Compression-Compressing-Deep-Neural-Networks-with-Pruning-Trained-Quantization-and-Huffman," May 2021, original-date: 2018-06-28T19:38:06Z. [Online]. Available: https://github.com/bemova/Deep-Compression-Compressing-Deep-Neural-Networks-with-Pruning-Trained-Quantizati

[66] C. Shangyu, "csyhhu/Awesome-Deep-Neural-Network-Compression," May 2021, original-date: 2019-02-28T04:32:44Z. [Online]. Available: https://github.com/csyhhu/Awesome-Deep-Neural-Network-Compression

[67] T.-D. Truong, V.-T. Nguyen, and M.-T. Tran, "Lightweight Deep Convolutional Network for Tiny Object Recognition:," in *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018. doi: 10.5220/0006752006750682. ISBN 978-989-758-276-9 pp. 675–682. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006752006750682

[68] "Classification accuracy (in percent) comparison of CNN-Based models." [Online]. Available: https://www.researchgate.net/figure/Classification-accuracy-in-percent-comparison-of-CNN-Based-models_tbl3_336061238

[69] "FIGURE 1: Ball chart reporting the Top-1 and Top-5 accuracy vs...." [Online]. Available: https://www.researchgate.net/figure/

Ball-chart-reporting-the-Top-1-and-Top-5-accuracy-vs-computational-complexity-Top-1-and_fig1_328509150

Appendix A

Model architectures

Table A.1: Model architecture accuracies

Model	Model size (MB)	Top-Acc@1	Top-Acc@5
ResNeXt-101-32x8d	356.2	79.312	94.526
Wide ResNet-101-2	508.3	78.848	94.284
Wide ResNet-50-2	275.9	78.468	94.086
ResNet-152	241.7	78.312	94.046
ResNeXt-50-32x4d	100.5	77.618	93.698
ResNet-101	178.8	77.374	93.546
Inception v3	109	77.294	93.45
Densenet-161	115.9	77.138	93.56
Densenet-201	81.4	76.896	93.37
ResNet-50	102.5	76.13	92.862
Densenet-169	57.6	75.6	92.806
Densenet-121	32.5	74.434	91.972
VGG-19 with batch normalization	574.8	74.218	91.842
MobileNet V3 Large	22.1	74.042	91.34
MNASNet 1.0	17.8	73.456	91.51
VGG-16 with batch normalization	553.5	73.36	91.516
ResNet-34	87.3	73.314	91.42
VGG-19	574.7	72.376	90.876
MobileNet V2	14.3	71.878	90.286
VGG-16	553.4	71.592	90.382
VGG-13 with batch normalization	532.3	71.586	90.374
VGG-11 with batch normalization	531.5	70.37	89.81
VGG-13	532.2	69.928	89.246
GoogleNet	52.2	69.778	89.53
ResNet-18	46.8	69.758	89.078
ShuffleNet V2 x1.0	9.3	69.362	88.316
VGG-11	531.5	69.02	88.628
MNASNet 0.5	9.1	67.734	87.49
MobileNet V3 Small	10.3	67.668	87.402
ShuffleNet V2 x0.5	5.6	60.552	81.746
SqueezeNet 1.1	5	58.178	80.624
SqueezeNet 1.0	5	58.092	80.42
AlexNet	244.4	56.522	79.066

Table A.2: SqueezeNet architecture feature layers 1-9

Layer No	Layer	Layer size (KB)	Cumulative model size	Input size	Output size
FEATURES				16,3,224,224	
0	Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2))	8.2			16,64,111,111
1	ReLU(inplace=True)				16,64,111,111
2	MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)				16,64, 55, 55
3	**FIRE**			8.2	16,64, 55, 55
	(squeeze): Conv2d(64, 16, kernel_size=(1, 1), stride=(1, 1))	5.2			16,16,55,55
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))	5.4		16,16,55,55	16,64,55,55
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	38.1		16,16,55,55	16,64,55,55
	(expand3x3_activation): ReLU(inplace=True)				
		47.6			16,128,55,55
4	**FIRE**			55.3	16,128,55,55
	(squeeze): Conv2d(128, 16, kernel_size=(1, 1), stride=(1, 1))	9.3			16,16,55,55
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))	5.4		16,16,55,55	16,64,55,55
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	38.1		16,16,55,55	16,64,55,55
	(expand3x3_activation): ReLU(inplace=True)				
		51.7			16,128,55,55
5	MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)	0.5			16,128,27,27
6	**FIRE**			106.7	16,128,27,27
	(squeeze): Conv2d(128, 32, kernel_size=(1, 1), stride=(1, 1))	17.5			16,32,27,27
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))	17.9		16,32,27,27	16,128,27,27
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	149		16,32,27,27	16,128,27,27
	(expand3x3_activation): ReLU(inplace=True)				
		183.4			16,256,27,27
7	**FIRE**			290	16,256,27,27
	(squeeze): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1))	33.9			16,32,27,27
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))	17.9		16,32,27,27	16,128,27,27
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	149		16,32,27,27	16,128,27,27
	(expand3x3_activation): ReLU(inplace=True)				
		199.8			16.256,27,27
8	MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)	0.5			16,256,13,13
9	**FIRE**			489.6	16,256,13,13
	(squeeze): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1))	50.3			16,48,13,13
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))	38.6		16,48,13,13	16,192,13,13
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	333.5		16,48,13,13	16,192,13,13
	(expand3x3_activation): ReLU(inplace=True)				
		421.5			16,384,13,13

Table A.3: SqueezeNet architecture feature layers 10-12 and output layers

Layer No	Layer	Layer size (KB)	Cumulative model size	Input size	Output size
FEATURES				16,3,224,224	
10	**FIRE**		911.2	16,384,13,13	
	(squeeze): Conv2d(384, 48, kernel_size=(1, 1), stride=(1, 1))	74.9			16,48,13,13
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))	38.6		16,48,13,13	16,192,13,13
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	333.5		16,48,13,13	16,192,13,13
	(expand3x3_activation): ReLU(inplace=True)				
		446.1			16,384,13,13
11	**FIRE**		1357	16,384,13,13	
	(squeeze): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1))	99.6			16,64,13,13
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))	67.6		16,64,13,13	16,256,13,13
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	591.8		16,64,13,13	16,256,13,13
	(expand3x3_activation): ReLU(inplace=True)				
		758			16,512,13,13
12	**FIRE**		2115	16,512,13,13	
	(squeeze): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))	132.3			16,64,13,13
	(squeeze_activation): ReLU(inplace=True)				
	(expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))	67.6		16,64,13,13	16,256,13,13
	(expand1x1_activation): ReLU(inplace=True)				
	(expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))	591.8		16,64,13,13	16,256,13,13
	(expand3x3_activation): ReLU(inplace=True)				
		790.7			16,512,13,13
Classifier Layers			2906	16,512,13,13	
	Dropout(p=0.5, inplace=False)				16,512,13,13
	Conv2d(512, 1000, kernel_size=(1, 1), stride=(1, 1))	2053			16,1000,13,13
	ReLU(inplace=True)				16,1000,13,13
	AdaptiveAvgPool2d(output_size=(1, 1))				16,1000,1,1
		2053			
			4960		(16, 1000)
Eye tracking classifiers	AdaptiveAvgPool2d(output_size=(1))				
	Dropout(p=0.2, inplace=False)				
	Flatten				
Final regression layer	Linear(512, 25)				
	Denormalize(mean,std)				(1, 25)

For DIVA

{
"Author1": {
},
"Degree": {"Educational program": ""},
"Title": {
"Main title": "",
"Language": "eng" },
"Alternative title": {
"Main title": "",
"Language": "swe"
},
"Other information": {
"Year": "2023", "Number of pages": "vi,61"}
}