University of Piraeus

MSc Big Data & Analytics
Department of Digital Systems

Master Thesis

DATA PROCESSING FROM SENSORS AT THE EDGE

Konstantinos Pagkos

Supervisor:
Ilias Maglogiannis, Professor

Piraeus

15/6/2023

Master Thesis

Data Processing from Sensors at the Edge

Konstantinos Pagkos

A.M.: ME 2029

ABSTRACT

The purpose of this thesis is to test and examine the capabilities of different state-of-the-art convolutional neural network architectures for edge applications and evaluate their maturity for use in real-time medical applications. For this reason, we utilize a publicly available annotated dataset containing images of the human gastrointestinal tract and the use of one of the most advanced AI edge accelerators. Ultimately, we test, evaluate and compare the performance of several models and provide insight both into the nature of the dataset, as well as into the capacity and potential of the latest advancements on the field of lightweight convolutional neural networks optimized for embedded devices.

KEYWORDS: Convolutional Neural Networks, Medical Image Processing, Edge Applications

Acknowledgments

First of all, I would like to express my appreciation to my thesis supervisor, Professor Ilias Maglogianis, without whose advice and guidance, this endeavor would be impossible. I would also like to thank my family for the support and the encouragement they showed me all these years. Last but not least, I want to thank the growing community of Data Scientists and Machine Learning Engineers out there, who so kindly offer their knowledge and experience to those following in their footsteps.

Contents

1. Introduction 11
   1.1 Motivation 11
   1.2 Diseases of the Gastrointestinal Tract 12
   1.3 Healthcare Edge Applications 12
   1.4 Structure of the Thesis 13
2. Dataset & Related Work 14
   2.1 Kvasir v2 Dataset 14
      2.1.1 Anatomical Landmarks 14
      2.1.2 Pathological Findings 15
      2.1.3 GI Procedures 16
   2.2 Related Work 17
3. Background Theory 19
   3.1 Machine Learning 19
   3.2 Deep Learning & Artificial Neural Networks 19
      3.2.1 Activation Functions 20
   3.3 Deep Learning for Computer Vision 22
      3.3.1 Convolutional Layers 23
      3.3.2 Pooling Layers 25
      3.3.3 Fully Connected Layers 26
      3.3.4 Batch normalization 26
   3.4 Artificial Neural Network Training 27
      3.4.1 Loss Function 28
      3.4.2 Optimizer 28
      3.4.3 Regularization 28
      3.4.4 Transfer Learning 30
   3.5 Medical Image Processing 30
   3.6 Tiny Machine Learning 31
      3.6.1 Edge AI Accelerators 32
      3.6.3 Edge Optimization Techniques 34
4. Convolutional Neural Networks for Edge Applications 38
   4.1 MobileNetV2 38
      4.1.1 Inverted Residuals 38
      4.1.2 Linear Bottlenecks 39
      4.1.3 Architecture 39
   4.2 GhostNet 40

        4.2.1 Feature Map Redundancy and the GhostNet Module 40
        4.2.2 Architecture 42
        4.2.3 GhostNetEdgeTPU 43
    4.3 MobileNetEdgeTPU V2 43
        4.3.1 Fused Inverted Bottlenecks 43
        4.3.2 Architecture 44
    4.4 EfficientNet-Lite 45
        4.4.1 Compound Coefficient Scaling 45
        4.4.2 EfficientNet-Lite 46
    4.5 Ensemble Models 47
        4.5.1 Ensemble Model 1 47
        4.5.2 Ensemble Model 2 48
5. Methodology 50
    5.1 Data Pre-processing 50
        5.1.1 Split Images 50
        5.1.2 MAPGI Framework 50
        5.1.3 Resize & Rescale 53
        5.1.4 Image Augmentation 53
    5.2 Training Configuration 54
        5.2.1 Training Setup 54
        5.2.2 Training Parameters 54
    5.3 Post Training Optimization 56
        5.3.1 Post Training Quantization 56
        5.3.2 Compilation for the Edge TPU processor 56
    5.4 Inference 57
    5.5 Evaluation 57
        5.5.1 Metrics 57
        5.5.2 Explainability 59
6. Results 61
    6.1 Model Results 61
    6.2 Discussion & Conclusions 63
    6.3 Future Work 69
APPENDIX 70
    Training Curves 70
References 72

List of Figures

Figure 1. Samples from the Z-line class 14
Figure 2. Samples from the Pylorus class 15
Figure 3. Samples from the Cecum class 15
Figure 4. Samples from the Esophagitis class 15
Figure 5. Samples from the Polyps class 16
Figure 6. Samples from the Ulcerative Colitis class 16
Figure 7. Samples from the Dyed & Lifted Polyps class 17
Figure 8. Samples from the Dyed Resection Margins class 17
Figure 9. Simplified Visualization of an ANN 20
Figure 10. Visualization of a neuron 20
Figure 11. Hyperbolic Tangent graph 21
Figure 12. Sigmoid graph 21
Figure 13. ReLU graph 22
Figure 14. Layer visualization of a CNN 23
Figure 15. Visualization of the convolution operation 23
Figure 16. Feature map examples 24
Figure 17. Visualization of the depthwise convolution operation 25
Figure 18. Max pooling operation with 2x2 window and stride = 2 25
Figure 19. Batch Normalization during inference 27
Figure 20. Effects of dropout on an ANN 29
Figure 21. Google Coral Edge TPU Dev Board mini 33
Figure 22. Visualization of the effects of pruning 35
Figure 23. 32 Float format 36
Figure 24. Weight clustering visualization 37
Figure 25. Conventional residual Block 38
Figure 26. MobileNetV2 residual block 39
Figure 27. MobileNetV2 bottlenecks 39
Figure 28. MobileNetV2 architecture 40
Figure 29. GhostNet feature maps 41
Figure 30. Ghost module 42
Figure 31. Ghost Bottlenecks 42
Figure 32. GhostNet architecture 43
Figure 33. Inverted Bottleneck variants 44
Figure 34. MobileNetEdgeTPUV2 performance comparison on Pixel 6 CPU 45
Figure 35. Single scaling options vs. Compound Scaling 46
Figure 36. EfficientNetB0 architecture 47
Figure 37. Building blocks of ensemble model 1 48
Figure 38. Building blocks of ensemble model 2 49
Figure 39 MAPGI framework's steps: (a) Masking, (b) Crop, (c) MAVGA, (d) Lowpass Filter, (e) Resize & Rescale 51
Figure 40. Image Augmentation example 53
Figure 41. Edge TPU compiler report 56
Figure 42 Ensemble Model 2 Confusion Matrix 63
Figure 43 MobileNetV2 Confusion Matrix 64
Figure 44. Vanilla Saliency map samples for each class 65
Figure 45. Vanilla Saliency map samples for Pylorus Class……………………………..66
Figure 45. Vanilla Saliency map samples for Z-Line Class……………………………....66
Figure 45. Vanilla Saliency map samples for Esophagitis Class………………………....67

List of Tables

Table 1. Comparison of previous work 18
Table 2. Dev Board mini specifications...33
Table 3 Training Parameters 55
Table 4. Model results 61
Table 5. Quantized model results 61
Table 6. Model storage and computational requirements 62
Table 7. Model inference time on the Edge TPU 62

List of Abbreviations

AI	Artificial Intelligence
GI	Gastrointestinal Tract
FPS	Frames per Second
CNN	Convolutional Neural Network
TPU	Tensor Processing Unit
IBS	Irritable Bowel Syndrome
IBD	Inflammatory Bowel Disease
WHO	World Health Organization
VV	Vestre Viken Health Trust
CRN	Cancer Registry of Norway
EMR	Endoscopic Mucosal resection
ML	Machine Learning
ANN	Artificial Neural Network
Tanh	Hyperbolic tangent
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
DL	Deep Learning
TinyML	Tiny Machine Learning
MAPGI	Modular Adaptive Pre-processing for GI tract images
MAVGA	Mean-Approximated Gamma Value Adjustment
CLAHE	Contrast-Limited Adaptive Histogram Equalization

OS	Operating System
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
MDT	Mendel Development Toolkit
FPS	Frames per Second
FLOPS	Floating-Point Operations per second
SE	Squeeze – Excitation
NAS	Neural Architecture Search
IBN	Inverted Bottleneck
GC-IBN	Group Convolution-based IBN
SOC	System-on-Chip
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
MCC	Mathew’s Correlation Coefficient
CVPR	Conference on Computer Vision and Pattern Recognition

1. Introduction

1.1 Motivation

The last decades we have experienced a sudden and rapid advancement in the field of Artificial Intelligence. Machine Learning and Deep Learning applications have touched and transformed many aspects of technology, from simple daily utilities to cutting edge development.

One of the most promising and challenging fields for Machine Learning applications, has proven to be the medical one. Despite a heavy focus on this field from the research community, there is still a lot of room for improvement on Healthcare Machine Learning applications, mainly due to obstacles like the difficulty of gathering data annotated by certified practitioners, maintaining data privacy and a zero-error tolerance policy.

Video and capsule endoscopy is a particular subfield that includes procedures where, either the practitioners insert a flexible fiber optic cable through cavities and skin incisions, or the patient's shallow vitamin-size capsules equipped with cameras, in order to visually examine inner parts of the human body. Examples of such operations include laparoscopic procedures performed by surgeons, intubations performed by anaesthesiologists, colonoscopies etc. All these procedures can significantly benefit from Computer Vision algorithms. Image classification models can be used prevent misdiagnosis, while object detection and image segmentation models can provide real time assistance to practitioners during the procedures.

Computational performance of the aforementioned algorithms, plays a significant role on real time procedures, e.x. laparoscopic surgeries, where a model's latency and inference time contribute in the overall performance as much as its accuracy. Furthermore, combining small and efficient Convolutional Neural Networks with embedded devices can greatly extend the capabilities of telemedicine and self-diagnosis applications.

While there is an extensive bibliography focusing on the development of Machine Learning models for classification and segmentation on images procured by endoscopic means, most of them focus mainly on achieving the best precision/accuracy possible, paying little to no consideration on the computational performance.

In the scope of this thesis, we aim to test and explore the advancements on the field of embedded computer vision and how the development of edge computing Machine Learning applications, capable of processing endoscopic images with low latency and high Frames Per Second (FPS) throughput, can be utilized with the aim of providing assistance to the medical practitioners, facilitating telemedicine applications and ultimately reduce human errors and misdiagnosis. Specifically, we evaluate different Convolutional Neural Network (CNN) architectures on their performance on automatic disease detection of the Gastrointestinal Tract and their ability to run inference efficiently on Google's state-of-the-art Edge Tensor Processing Unit (TPU) accelerator.

1.2 Diseases of the Gastrointestinal Tract

Diseases, abnormalities and pathological findings of the Gastrointestinal (GI) Tract refer to conditions involving the esophagus, stomach, small / large intestine, rectum and the accessory organs of digestion, the liver, gallbladder and pancreas. Some common examples of GI diseases include peptic ulcers, irritable bowel syndrome (IBS), inflammatory bowel disease (IBD), such as Crohn's disease and ulcerative colitis, as well as gastrointestinal cancers, such as stomach and colorectal cancer. Symptoms of GI diseases can vary depending on the condition and have a significant impact on a person's quality of life. According to the World Health Organization (WHO), GI diseases are a major public health problem worldwide. Gastrointestinal cancers are the third leading cause of cancer deaths globally, responsible for around 7.2 million deaths per year. In addition, IBD affects an estimated 3 million people in the United States alone and it's estimated to affect more than 5 million people worldwide. Furthermore, IBS is a common condition that affects up to 20% of the population in developed countries.

As most of these diseases can be cured, proper diagnosis and treatment are crucial for managing symptoms and preventing life threatening situations. For this reason, there has been an extensive bibliography of computer vision applications aiming to classify medical images produced by endoscopic means. These applications can identify patterns and markers associated with different diseases and as a result aid in the early detection of conditions such as polyps and tumors, which can lead to the diagnosis and treatment of gastrointestinal cancers.

1.3 Healthcare Edge Applications

Edge computer vision technology has the potential to play a key role in preventing and managing diseases of the gastrointestinal tract in remote or resource-limited settings. Edge computing is a technology that allows data analysis to happen at or near the source, rather than relying on transmitting it to a central location for processing. This enables real-time data diagnosis and decision-making at the point of care, which can be especially important in situations where internet connectivity or resources are limited.

For example, a portable endoscope equipped with edge computer vision technology could enable a healthcare practitioner to capture images of the gastrointestinal tract and perform real-time diagnosis on remote locations. In the same manner, combining the images produced from capsule endoscopy with low computational complexity computer vision applications that can run on mobile devices, can effectively enable self-diagnosis in remote settings where access to specialized medical equipment and personnel may be limited. The aforementioned applications can be further utilized in post-operative monitoring, enabling doctors to track the healing process and detect any complications early on, without requiring their physical presence.

1.4 Structure of the Thesis

Chapter 1 presents the scope of the thesis, its goals and the motivation behind it. In **Chapter 2** we introduce the Kvasir v2, a dataset comprised of annotated images of the gastrointestinal tract, as well as the results of previous works found in literature that utilize it. **Chapter 3** contains all the background theory related to the main topics revolving around the thesis. This includes the basic concepts of Deep Learning, convolutional neural networks, image processing and edge applications. **Chapter 4** goes into detail about the architecture of different lightweight convolutional neural networks suitable for edge applications. **Chapter 5** describes the methodology, specifically the end-to-end steps of the process used in this thesis, from data processing all the way to the evaluation of the models. Finally, **Chapter 6** contains all the results from the experiments, the discussion and proposals for future work.

2. Dataset & Related Work

2.1 Kvasir v2 Dataset

In this thesis we have used the Kvasir v2 dataset, a multi-class image dataset for computer aided gastrointestinal disease detection. The dataset consists of a total of 8.000 medical images obtained through the means of endoscopic procedures in the gastrointestinal tract. The images have been collected with the use of appropriate equipment from the Vestre Viken Health Trust (VV) in Norway, which consists of four different hospitals that tend to the medical needs of 470.000 patients. Each image has been annotated and verified by certified medical professionals either from the VV or the Cancer Registry of Norway (CRN).

The dataset consists of eight different classes, containing 1.000 images each. The resolution of these images differs, starting from 720x576 pixels up to 1920x1072. Some of the images also contain a small green box in the lower left corner, depicting the position of the endoscope inside the tract.

Below the eight different classes consisting this dataset are briefly presented:

2.1.1 Anatomical Landmarks

Anatomical landmarks are parts of the digestive tract that can be used as points of reference during endoscopic procedures.

The **Z-line** marks the transition site between the esophagus and the stomach. This transition is made visible by the change in colour, where the white esophageal mucosa turns into the gastric red mucosa of the stomach. This border is of great interest, as many signs of disease can be detected here.

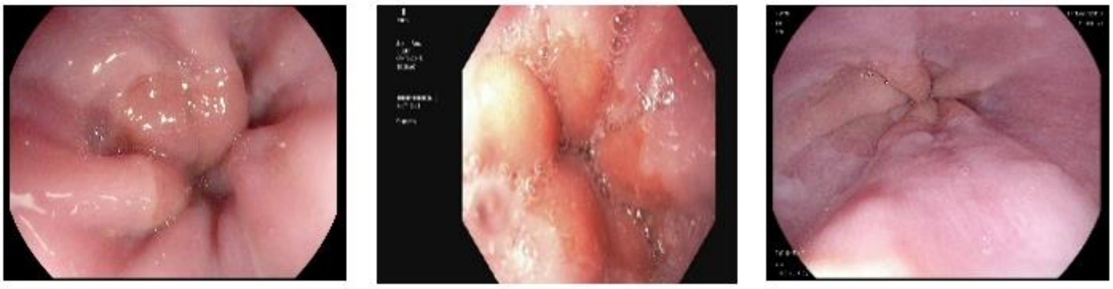


Figure 1. Samples from the Z-line class

The **Pylorus** is the small opening connecting the stomach with the first part of the small bowel. Correctly identifying the pylorus can be of assistance for maneuvering endoscopic equipment.

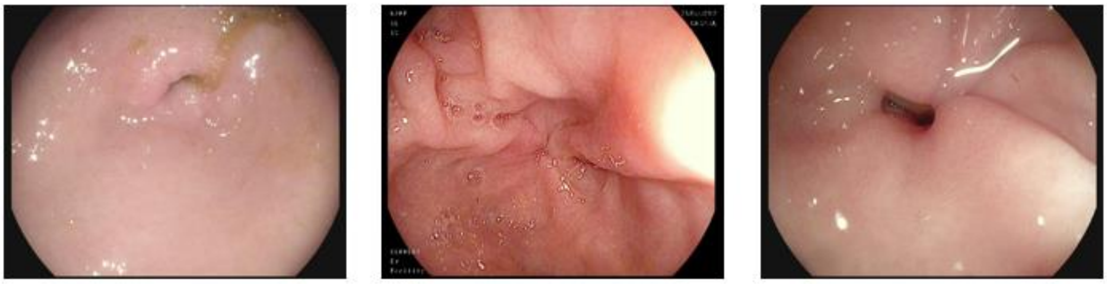


Figure 2. Samples from the Pylorus class

The **Cecum** is a pouch within the peritoneum that is considered to be the start of the large intestine. This point marks the completion of a colonoscopy.

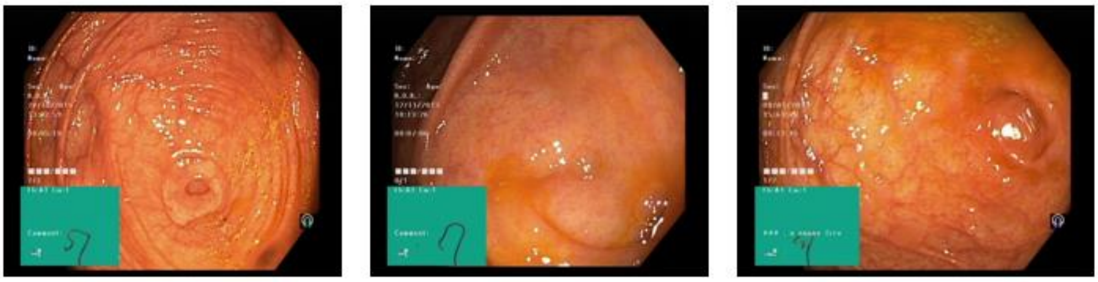


Figure 3. Samples from the Cecum class

2.1.2 Pathological Findings

These findings refer to a number of conditions and diseases that can be visually detected in the GI tract.

Esophagitis is essentially inflammation of the esophagus, presented in the form of visible breaks in the mucosa, which are created when gastric acid flows back from the stomach due to certain conditions. The extend and length of the breaks can be used in order to determine the severity.

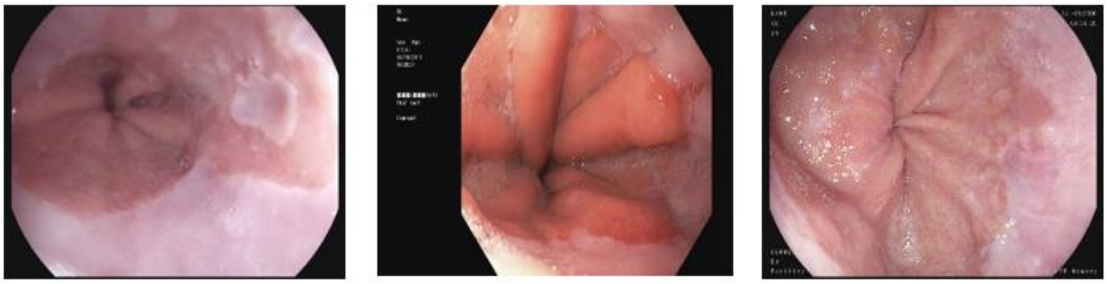


Figure 4. Samples from the Esophagitis class

Polyps are lesions detected within the bowel in the form of outgrowing mucosa. Their size and shape can vary, but can be detected by inspecting thir colour and texture. Polyps more often than not are harmless, but in cases they can lead to colorectal cancer if note removed.

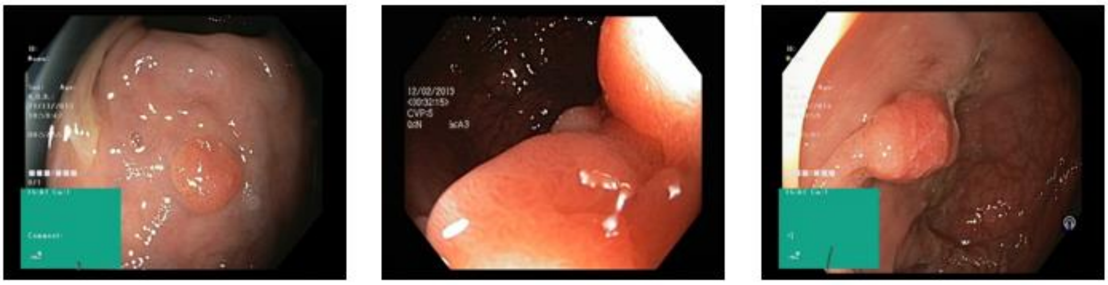


Figure 5. Samples from the Polyps class

Ulcerative Colitis is a chronic inflammatory disease located within the large bowel. The severity of the disease can mild, made visible by swollen red areas, but in extreme cases it can lead to the development of ulcerations, negatively affecting the quality of life of the patient.

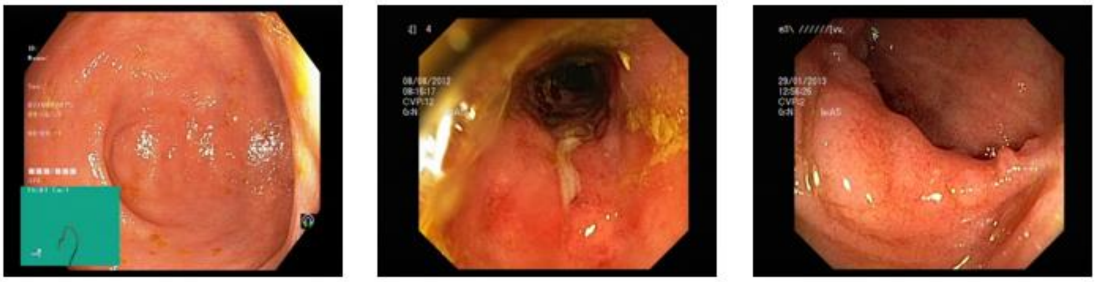


Figure 6. Samples from the Ulcerative Colitis class

2.1.3 GI Procedures

The procedure of interest is called endoscopic mucosal resection (EMR) and is one of the most common polyp removal techniques. During this procedure, liquid is injected underneath the polyp in order to separate it from the underlying mucosa. The solution may also contain a staining dye, in order to highlight the polyp's borders. The elevated polyp is then extracted.

Dyed & Lifted Polyps are a visual example of a polyp after a coloured solution is applied. The margins between the polyp and the underlying tissue are clearly visible.

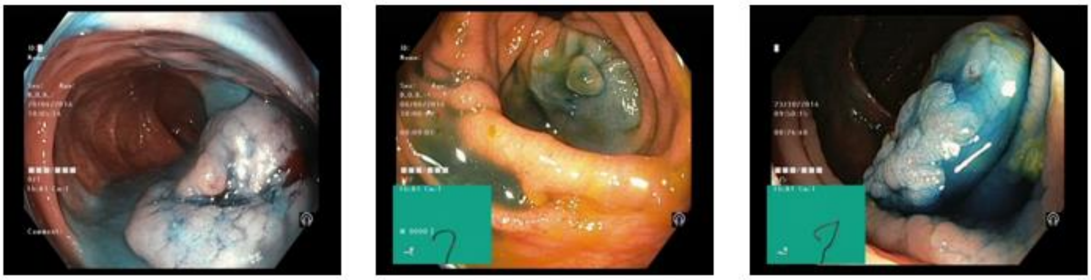


Figure 7. Samples from the Dyed & Lifted Polyps class

Dyed Resection Margins refer to the area of interest, after the polyp has been extracted. Potential residual polyp tissue must be detected and accordingly removed.

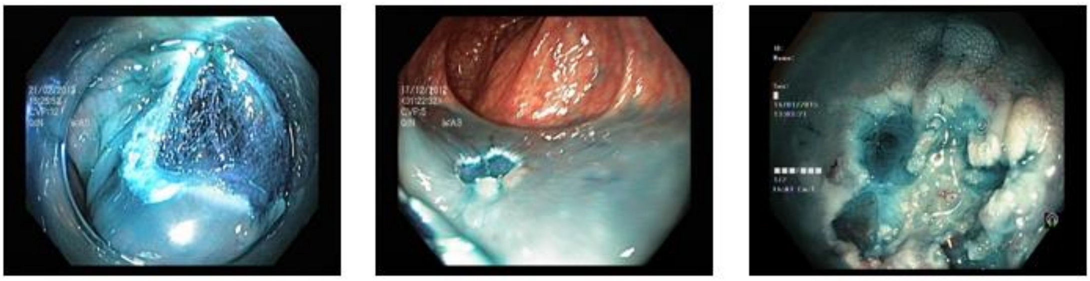


Figure 8. Samples from the Dyed Resection Margins class

2.2 Related Work

Taking into account the difficulties of the data collection procedure in the medical field and the fact that the Kvasir v2 dataset is not only one of the most complete annotated image datasets of the GI tract, but also publicly available, it is only logical that over the years there has been an extensive variety of deep learning and machine learning based approaches making use of it. For the sake of simplicity, we will use accuracy as the base metric for comparing them.

Starting at 2017, when Pogorelov et al. [9] introduced the Kvasir v2 dataset, essentially doubling the sample size of the original Kvasir. Along with the data, they also experimented with 8 different classification approaches in order to establish a baseline for future researchers. These approaches included different CNNs and also other supervised classification methods, with the best performing being a 3-layer CNN, achieving 95.9% accuracy.

Following their work, researchers experimented on the Kvasir v2 dataset using state of the art CNN architectures available at the time. These architectures included Resnet50 [24], GoogleNet [8], ResNet-18 [8], DenseNet-201 [7] MobileNetV2 [18] etc. As most of these models struggled to surpass the performance established by the original paper, modified architectures of the same CNNs, i.e., modified VGG-16 [8], and custom architectures with larger input size [11] were introduced, that seemed to fare better in terms of accuracy.

Another approach followed by [7] and [18] is the ensemble learning methods, combining a number of the aforementioned CNNs as features extractors and then retraining a classifier on the new feature maps. This approach yielded the best results, where [18] scored the best accuracy of value 99.2%.

Last but not least, [26] focused on improving the classifier instead of the feature extractor, comparing the performance of the commonly used soft-max layer against a Support Vector Machine (SVM) and a Stacked LSTM classifier, and [25] used Capsule Networks based on DenseNet-121.

Even though the metrics achieved by many approaches are considered more than sufficient, the limitations of previous works is that the researchers did not take heavily into account the computational complexity of their approaches. Most of the models proposed consist of tens of millions of parameters, rendering them unsuitable for applications at the edge. In order to further extend their work, we aim to explore the trade-off between model size and accuracy on the same dataset.

Base Model Architecture	**Top 1 Accuracy**	**Parameters**	**Reference**
3-layer CNN	*92.40*	*~23.000.000*	[9]
ResNet50	*95.70*	*~25.600.000*	[24]
Proposed CNN	*96.80*	*~2.666.312*	[11]
Ensemble Method	*97.38*	*>20.000.000*	[7]
Modified VGG16	*96.33*	*>100.000.000*	[8]
Proposed Attention Model	*92.84*	*19.920.000*	[16]
DenseNet121+I/O Modules	*94.82*	*~*	[25]
ResNet50	*91.40*	*~25.000.000*	[18]
MobileNetV2	*88.00*	*~2.600.000*	[18]
Xception	*97.04*	*~22.000.000*	[18]
Ensemble Method	*99.29*	*>50.000.000*	[18]
Ensemble Method + Stacked LSTM classifier	*97.90*	*>70.000.000*	[26]

Table 1. Comparison of previous work

3. Background Theory

3.1 Machine Learning

Machine Learning is a specific branch of Artificial Intelligence that in the past few decades, came to challenge traditional programming. Through Machine Learning, we can build “intelligent machines” that are able to discover complex patterns in data, derive rules otherwise invisible to the human and ultimately make decisions and forecasts. The main difference of ML applications to traditional programming, is that the output of the latter is the result of a very specific set of rules described in detail by human programmers, where ML applications are able to “learn” and improve without human supervision.

Typically, ML algorithms are divided into three main categories:

- **Supervised learning**, usually involves models used in classification and regression tasks. In order to train these models, a labeled training dataset example needs to be provided.
- **Unsupervised learning**: most common examples are clustering and association algorithms. These algorithms are used to explore and draw inferences describing hidden structures from unlabeled data.
- **Semi-supervised learning**: models of this family are between supervised and semi-supervised, utilizing both labeled and unlabeled data.

3.2 Deep Learning & Artificial Neural Networks

Deep Learning is a subfield of Machine Learning, and in extend of Artificial intelligence. Deep Learning is associated with Artificial Neural Networks (ANNs), a complex group of algorithms loosely modeled after the human brain. The characterization Deep derives from the fact that ANNs are usually comprised from input / output layers and large number of hidden layers, able to discover and extract features from the data in a hierarchical way.

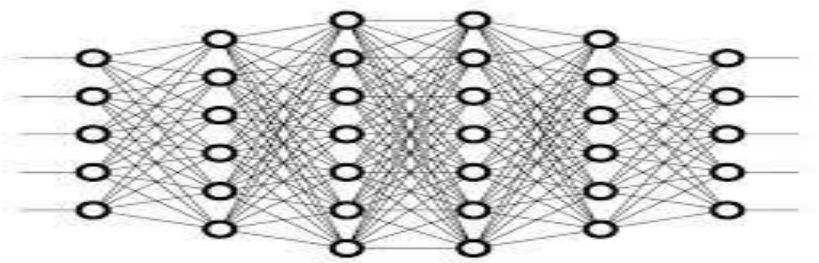


Figure 9. Simplified Visualization of an ANN

The building block of every ANN is the perceptron, that resembles a neuron. A single perceptron is a multiple input – single output algorithm that can be described from the following linear equation:

$$output = \sum_{i=0}^{n} w_i x_i + b \tag{1}$$

where, w is the weight, b is the bias and x is the input.

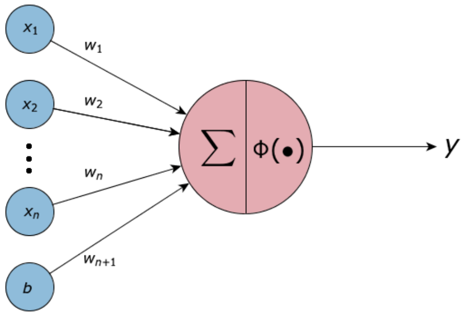


Figure 10. Visualization of a neuron

3.2.1 Activation Functions

An ANN is built from multiple layers, and each of these layers, from multiple perceptrons. Since the perceptron's equation is linear, a combination of multiple perceptrons would still remain linear, thus prohibiting the network from simulating more complex algorithms. For this reason, the activation function is introduced.

The activation functions practically allow the neural network to learn non-linear relations. With its implementation, a neuron's equation transforms as follows:

$$output = \varphi * \left(\sum_{i=0}^{n} w_i x_i + b \right) \qquad (2)$$

Where φ , is the activation function

There multiple activation functions. Here we present some of the most commonly used, their equations and their respective diagrams.

Hyperbolic Tangent (tanh):

$$\varphi_x = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (3)$$

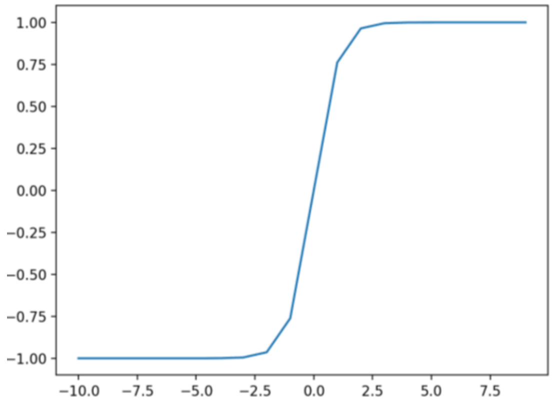


Figure 11. Hyperbolic Tangent graph

Sigmoid:

$$\varphi_x = \frac{1}{1 + e^{-x}} \qquad (4)$$

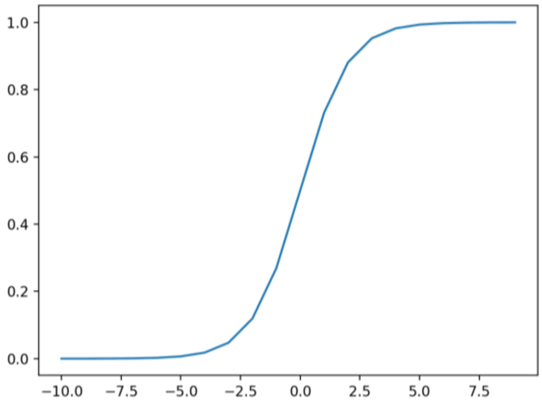


Figure 12. Sigmoid graph

Rectified Linear Unit (ReLU):

$$\varphi_x = \max(0, x) \qquad (5)$$

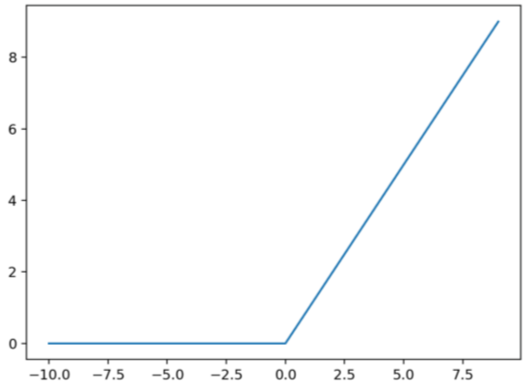


Figure 13. ReLU graph

Softmax is a very specific activation function, that combines all of its inputs and presents the same number of ouputs. It is commonly used at the last layer of an ANN and specifically in the task of classification. This function’s outputs lie in the range [0,1] and resemble probabilities, and consequently their sum equals 1.

$$\varphi_{x_i} = \frac{e^{x_i}}{\sum_j e^{xj}} \qquad (6)$$

3.3 Deep Learning for Computer Vision

Convolutional Neural Networks, or CNNs, are a class of artificial neural networks that are specifically used to extract features and patterns from signals, such as images, speech and audio. CNNs utilize multiple convolution layers in order to hierarchically extract features from the input. Earlier layers of the algorithms focus on simple features, such as edges and colors, whereas later convolutions are able to recognize larger objects and shapes and ultimately identify objects. CNNs advantage over other neural network architectures in computer vision tasks, is that the convolution takes into account the spatial correlation between pixels in an image.

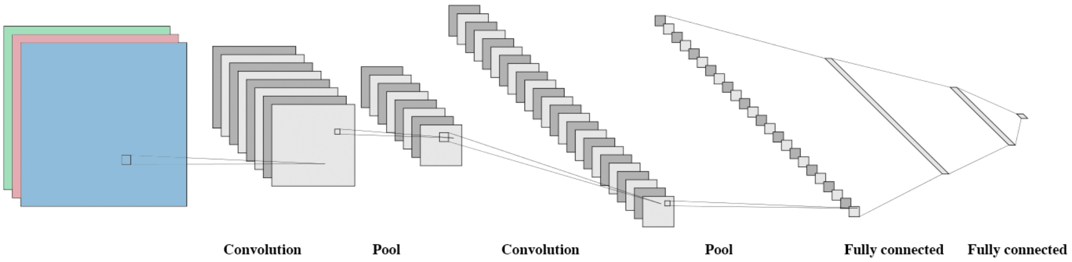


Figure 14. Layer visualization of a CNN

3.3.1 Convolutional Layers

The main building layer of CNN architectures is the Convolution Layer. During the act of convolution, we use a set of weighted matrices called kernels, or filters, to extract information from an image. The width and height of a kernel is smaller than the input's dimensions. In order to obtain the output, we iterate the kernel over every pixel in an image, starting from one side (i.e., top left). The output's width and height depend on the input's dimensions, the kernel's dimensions, as well as the convolution stride and padding size. The stride describes how much a filter will be shifted over the image after each convolution operation, while padding is a technique that extends the dimensions of the original image in order to more efficiently detect patterns at the edges.

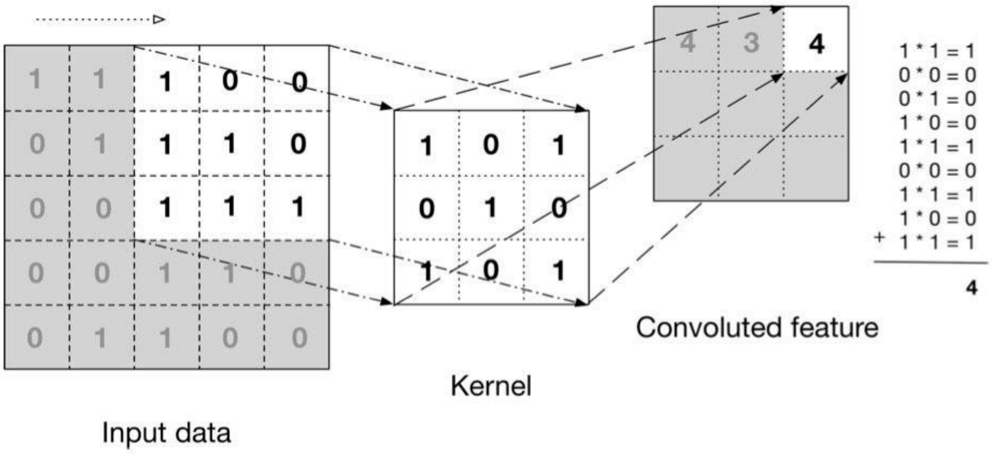


Figure 15. Visualization of the convolution operation

Accounting for all the parameters, given an image $W_{in} \times H_{in} \times C_{in}$, where W_{in},H_{in},C_{in} are the width, height and number of channels of the input image, the output $W_{out} \times H_{out} \times C_{out}$ of the convolution layer can be computed as:

$$W_{out} = \frac{(W_{in} + 2P - K_W)}{S_W} + 1 \tag{7}$$

$$H_{out} = \frac{(H_{in} + 2P - K_H)}{S_H} + 1 \quad (8)$$

$$C_{out} = K_C \quad (9)$$

Where K_w,K_H,K_C are the width, height of the kernel and the number of filters respectively, P is the padding size, assuming same size padding in every direction, and S_W,S_H are the strides across the two dimensions. C_{out} is also referred to as the feature maps produced by a convolution layer.

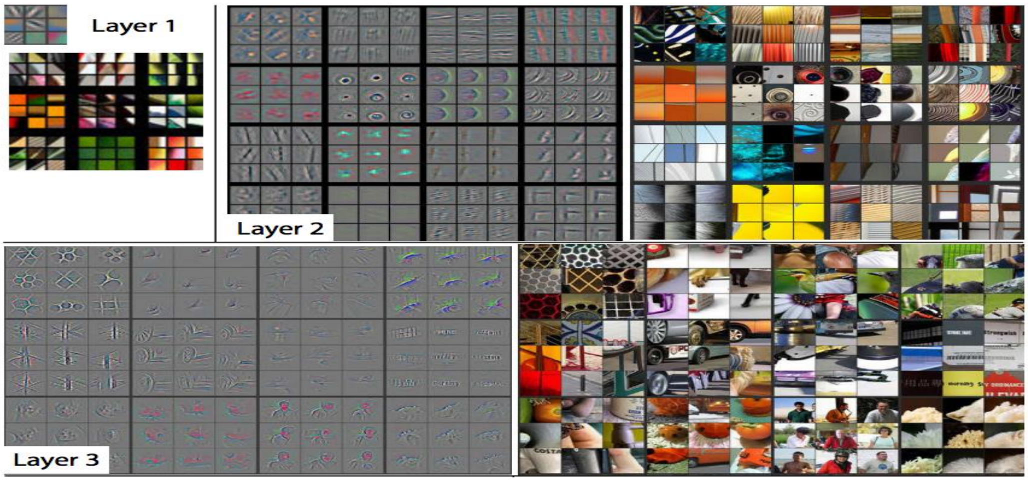


Figure 16. Feature map examples

In the scope of this thesis, we also mention a specific type of convolution called depthwise separable convolution, which can reduce the total parameters of a normal convolution layer more than 50%.

This is a two-step procedure. The first step is a depthwise convolution, which in contrast to normal convolution, convolves each input channel with a different filter and stacks the output feature maps.

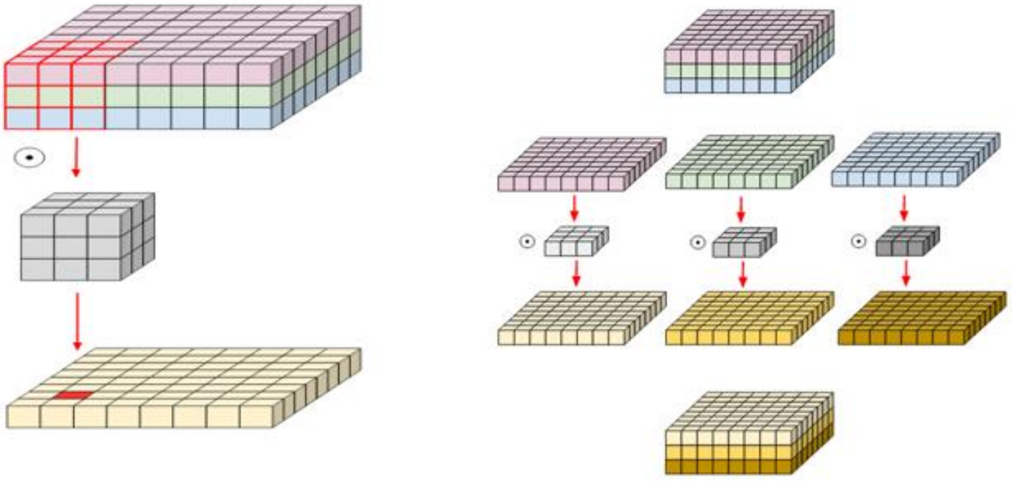


Figure 17. Visualization of the depthwise convolution operation

The second step is a normal convolution utilizing a 1x1 dimensional kernel in order to obtain the final output.

3.3.2 Pooling Layers

It is a quite common occurrence for a convolution layer to be followed by a pooling layer, even though modern architectures opt to use strided convolutions instead. The main purpose of a pooling layer is to compress the activation maps and reduce the number of parameters between the layers. Pooling layers typically have two parameters, kernel size and step, that work in the same manner in the convolution layer.

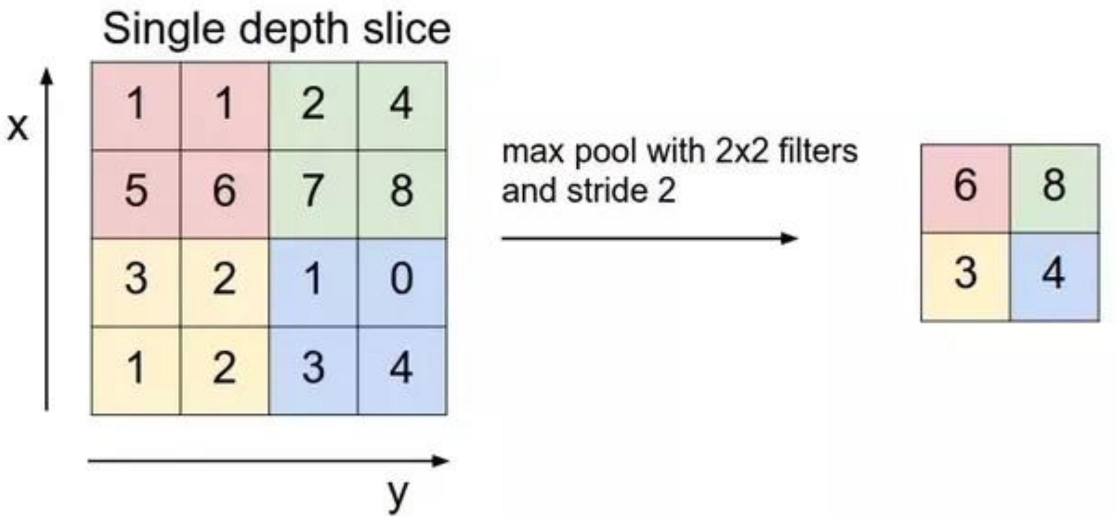


Figure 18. Max pooling operation with 2x2 window and stride = 2

The most commonly used pooling layers are max pooling and average pooling. Max pooling works by sliding a window over the input data and selecting the largest value of the window as output, while average pooling calculates the average value of the parameters inside the window.

3.3.3 Fully Connected Layers

Fully connected layers are the most basic layers of any ANN architecture. They consist of a set of neurons, which are connected to every neuron of both the previous and the next layers. In a CNN, fully connected layers are typically used in the last few layers, after the desired features have been extracted from previous convolutional layers, in order to map these features with the output. Nowadays, these layers are also usually replaced by convolution layers.

3.3.4 Batch normalization

Batch normalization is a layer that has been an essential part of CNN architectures since it was introduced. Batch normalization layers essentially normalize the output of one hidden layer before it is inserted to the next. A layer of this kind has a total of four parameters, two learnable ones called beta and gamma, and two non-learnable called mean moving average and variance moving average. The layer works as follows:

1. For each input vector, calculate the mean and variance of the values in the mini-batch.
2. Normalize the above values, with zero mean and unit variance.
3. Shift and scale the values accordingly by multiplying by the factor gamma and adding the factor beta.
4. Calculate the exponential moving average after each iteration and save the final result after training has concluded. This result is then used during inference, where we have only one input and not a batch.

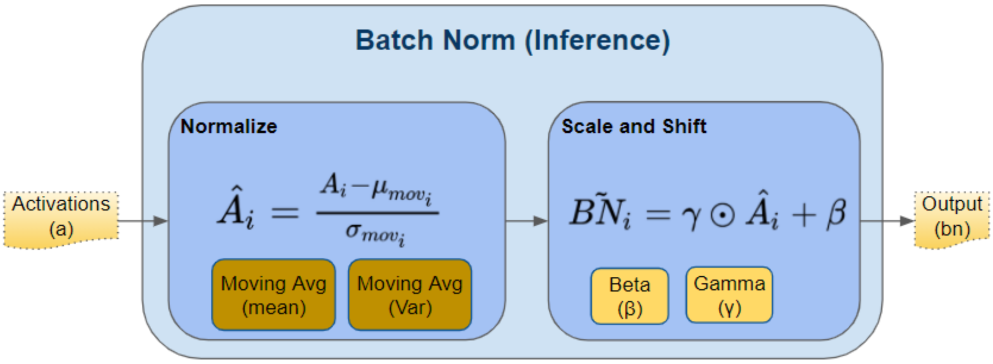


Figure 19. Batch Normalization during inference

The utilization of Batch Normalization layers after activation functions has shown significant improvement in weight convergence during training and subsequently reduction in total training time required.

3.4 Artificial Neural Network Training

Training may be the most important part in the successful implementation of an ANN. After choosing an appropriate ANN architecture for the task at hand, the network must be trained until it is capable of efficiently mapping the data. In order to achieve this, the weights and biases of every perceptron must be calibrated accordingly.

The training process can be briefly summarized in the following steps:

1. The data are presented to the network and an output is produce. This part is also called ***forward propagation***.
2. Compare the output with the ground truth and estimate the error with the use of a ***loss function***.
3. Using the quantified error and an ***optimizer***, calibrate the weights and biases of the network in order to minimize the error. This process is also called backward propagation
4. Repeat the above steps until there is no performance increase to be gained, measured by the appropriate metrics.

In the following section, the above concepts, also called ***hyperparameters*** of a neural network, are explained in depth.

3.4.1 Loss Function

Loss functions play one of the most important roles during the training of a model, that of evaluating its performance of the task at hand. Every step of every epoch, the loss function approximates the error between the model's output and the ground truth, with this error increasing the further the output deviates from the true label.

During backward propagation, we utilize the partial derivatives of the loss function in order to finetune the weights and biases of the whole neural network, using a chaining rule from right-to-left, or output-to-input.

One of the most common loss functions used for multiclass classification is the Cross-Entropy Loss. In order to use this function, the final layer of a model should contain the same number of nodes as the classes of the dataset and a softmax activation function. Cross-entropy will calculate a score, based on maximum-likelihood, that summarizes the average difference between the model's output probabilities and the ground truth. There is also the option of using class weights, where effectively the loss function "punishes" misclassifications of different labels with different weights.

3.4.2 Optimizer

The optimizer is basically an optimization scheme that during backpropagation, utilizes the partial derivatives of the loss function, or gradients, in order to properly update the existing weights and biases. Stochastic Gradient Descent (SGD) and its variants are the most used optimization algorithms.
SGD takes a step, called learning rate, towards the direction of the greatest descent for each weight. The learning rate is an important parameter for optimizing training. A higher learning rate might converge faster in contrast to a lower one, but might also miss a local minimum. For this reason, we usually utilize a learning rate schedule, where the training begins with a high learning rate, but it gradually decreases over time.

In addition to the learning rate, we also add another term called momentum, which basically determines how much impact the previous gradients should have on a certain weight. It achieves this by accumulating an exponentially decaying moving average of past gradients.

3.4.3 Regularization

With the term Regularization, we refer to a set of available techniques, which we can utilize in order to reduce the risk of overfitting and sometimes even improve model results. Below we present some of the most common techniques.

3.4.3.1 Dropout

Dropout is a regularization technique, that during a training epoch excludes, “drops out”, a random number of neurons from a layer. By temporarily removing a set of neurons from a layer, we essentially change the input and output connections of it. This process has the benefit of making a model more robust, because it effectively forces different nodes to assume more / less responsibility in their decisions, making each neuron less dependent from other neurons. Dropout has a proven record of improving the performance of neural networks.

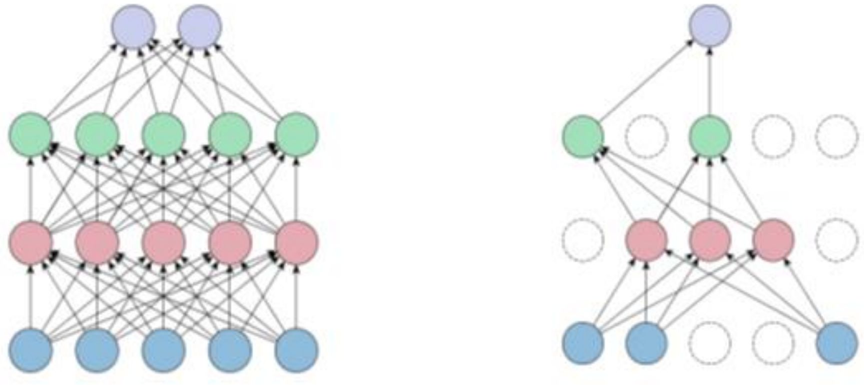


Figure 20. Effects of dropout on an ANN

3.4.3.2 L1 / L2 Regularization

L1 and L2 Regularization, also known as “weight decay”, is a way of preventing overfitting by adding an additional penalty to the prediction error of the loss function. This penalty comes in the form of absolute value of magnitude for L1 regularization, and squared magnitude for L2. The key difference between the two regularization techniques, is that the first shrinks the less important feature’s coefficient towards zero, effectively leading to sparsity.

3.4.3.3 Early Stopping

Early Stopping is the process of terminating the training of a neural network early, in order to avoid the deterioration of the training and validation losses, and ultimately overfitting. This technique, though simple, is widely used and it often produces networks that generalize better to the training data.

3.4.3.4 Data augmentation

As data augmentation, we define the process of creating new training data from the existing ones, by applying different transformations. These transformations vary, from simpler ones, e.x. rotating an image, to more advanced, like histogram equalization etc. Different use cases and data require different data augmentation techniques that match the task. Data augmentation is especially popular in Deep Learning applications, because more often than not, it is difficult to acquire new data in real world applications.

3.4.4 Transfer Learning

In real world applications, acquiring large volumes of data and training an ANN can prove expensive, both in terms of money and time. For this reason, we commonly used Transfer Learning, by using pre-trained models that have been trained on big datasets. There are many advantages in using Transfer Learning, such as:

1. Significant decrease in training time.

2. Increase in performance, because existing pretrained weights are more often than not better than random weight initialization.

3. Reduced risk of overfitting for small datasets, because during through Transfer Learning we can freeze some layers of the model and finetune the rest, or even freeze the whole model and use it as a feature extractor.

For Transfer Learning to be effective, both the data at hand and the data used to train the pretrained model should showcase at least some degree of similarity.

3.5 Medical Image Processing

Deep Learning applications have proved successful and further promising in various different fields, and the medical field is no exception. Massive amounts of medical data are produced daily worldwide and in many different forms, with images being one of the most common. Medical images are generated through a plethora of diagnostic procedures, like X-rays, magnetic resonance-imaging, tomographies etc. In addition to that there are number of procedures that utilize real time video applications, like laparoscopies.

Though the automatic processing of all these images using Deep Learning models is quite promising, the medical field also presents. some unique challenges. One challenge in particular that every developer has to account for, are the differences between similar images generated by different medical apparatuses. Even though these differences can be quite subtle, like variations in hue or contrast, if left unaddressed, they can prove critical in the successful operation of a model. There are several different techniques, i.e. contrast enhancement via gamma correction, that aims to adjust the relative brightness difference between objects and their backgrounds in images, in order to improve their visibility. These methods are an essential part in medical image processing, as they reduce the noise contained in the image and improve the robustness of CNNs, by countering low contrast occurring from inconsistent illumination and other factors.

Another common issue that derives from the nature of the field, is the no-error tolerance in the deployment of AI applications. Medical applications must provide a high degree of robustness and state-of-the-art performance, because errors in a model’s output can potentially have grave consequences. Due to this reason, until today, most applications are considered as a supporting tool for consulting by practitioners, instead of producing a definitive diagnosis.

Last but not least, another major challenge is the perseverance of the data privacy, due to the sensitive nature of an individual’s medical information. This fact can prove a significant barrier in the attempt to share data and create efficiently big datasets for such applications.

3.6 Tiny Machine Learning

The recent growth of AI applications in combination with the massive adoption of IoT devices have presented researchers with a new, challenging field, often referred to as Edge AI or Tiny Machine Learning (TinyML), By definition, TinyML is the deployment of AI applications in embedded devices, where the computations are performed close to the data source with minimal latency. These applications carry substantial benefits, such as real-time inference & insights, reduced costs and power consumption, increased privacy, high availability etc.

This kind of applications also present significant challenges. In order to deploy neural networks at resource constrained environments, one has to take into account multiple factors. Some of the main challenges are:

- The limited computational resources of embedded devices, in contrast with the high computational complexity of neural networks.
- The unpredictability that derives from continually receiving data through sensors from the real world.
- The high demand for robustness and resiliency

Due to these factors, there has been heavy focus on the research and development of efficient processors for edge AI and optimization techniques for neural networks. This effort can be broken down into two different approaches, creating efficient hardware accelerators and optimizing the architecture of neural networks. These explained below:

3.6.1 Edge AI Accelerators

In order to support the rise of TinyML applications, many organizations and manufacturers have focused on developing efficient hardware architectures and chips for Edge AI. These devices have only one purpose, to efficiently tackle the challenges of TinyML and bridge the gap between the high computational effort required by AI and the low processing capabilities of embedded microprocessors and integrated circuits. Along with many new startup companies, almost all major competitors in the semiconductor industry, like NVIDA, Google and Intel, have already shifted their focus and developed solutions for accelerating inference at the edge. Namely, some of the most popular solutions are listed below:

- NVIDIA Jetson series
- AMD EPYC Embedded series
- Intel’s Movidius Vision Processing Units
- Qualcom’s DM.2
- ARM Mali C-55
- Google Coral Edge TPU

In the scope of this thesis, we provide further information only on Google’s Edge TPU, and specifically the Google Coral Dev Board mini single-board computer.

3.6.1.1 Google Coral Dev Board mini

The Coral Dev Board mini is a single-board fully-functional embedded system that can be used as an evaluation and prototyping device for the Accelerator Module, a surface mounted module that incorporates an Edge TPU. The Edge TPU is capable of performing 4 tera-operations per second (TOPS), using 0.5 watts for each TOPS, making it one of the state-of-the-art ML accelerators on the edge.

The Board is also a fully-functional embedded system, featuring a Quad-core Arm Cortex Architecture and 2 GB RAM. The Board’s OS, Linux Mendel, in combination with the Wi-Fi access it offers, provides an easy-to-use development and testing environment. Furthermore, the Coral Team has provided the PyCoral API (Python), which provides an easier alternative to lower-level C++ libraries that are often required for such tasks.

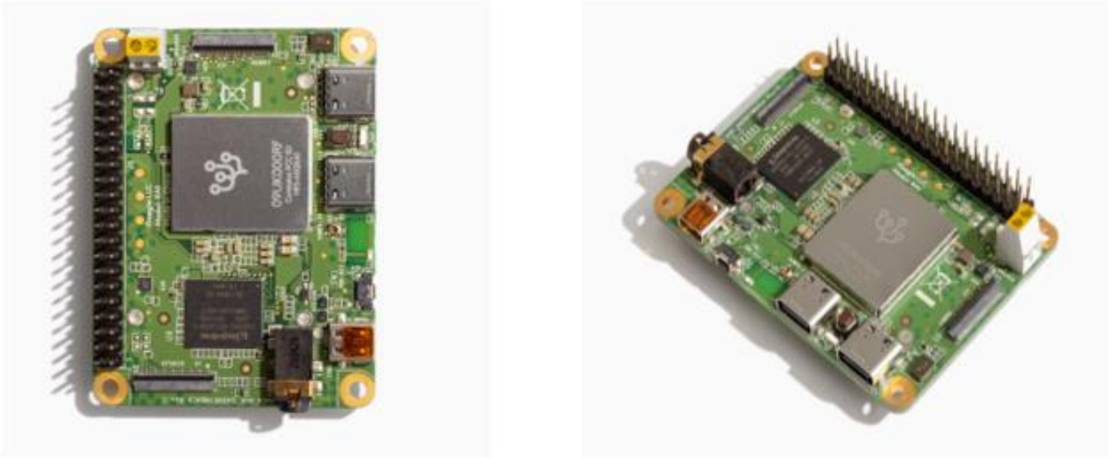


Figure 21. Google Coral Edge TPU Dev Board mini

Technical Specifications:

CPU	MediaTek 8167s SoC (Quad-core Arm Cortex-A35)
GPU	IMG PowerVR GE8300 (integrated in SoC)
ML Accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
RAM	2 GB LPDDR3
Wireless	Wi-Fi 5 (802.11a/b/g/n/ac); Bluetooth 5.0
Audio / Video	3.5mm audio jack; digital PDM microphone; 2.54mm 2-pin speaker terminal; micro HDMI (1.4); 24-pin FFC connector for MIPI-CSI2 camera (4-lane); 24-pin FFC connector for MIPI-DSI display
Flash Memory	8 GB eMMC
Input/Output	40-pin GPIO header; 2x USB Type-C (USB 2.0)

Table 2. Dev Board mini specifications

Software:

The board's OS is a lightweight Debian Linux variation, called Linux Mendel. This OS offers most of the commonly used Linux utilities. This fact, in addition to the board's hardware capabilities (Wi-Fi connection, 2GB RAM, enough space to store the validation dataset into the board), enables the developers to test their models in a fast and easy way.

The Mendel Development Tool (MDT) is a command line interface tool that allows easy access to devices running Mendel Linux, both for Windows 10 and Linux. After the board's initial setup, the MDT is used to gain instant access to the board's OS through WiFi, efficiently moving files from a PC to the board & vice versa.

PyCoral API

The PyCoral API is a small set of convenience functions that initialize the TensorFlow Lite Interpreter with the Edge TPU delegate and perform other inferencing tasks such as parse a labels file, pre-process input tensors, and post-process output tensors for common models. A Python script utilizing the PyCoral API is used for testing the models on the images already stored in the board.

Despite the Edge TPU's advantages, it comes with some constraints and/or drawbacks, such as:

- Models aiming to run inference on the Edge TPU, must be developed completely within TensorFlow's ecosystem, disabling the developer from using the vast selection of available models developed in PyTorch without significant effort.

- The Google Coral Edge TPU only supports specific operations (layers). Failure to meet these constraints results in a model that utilizes the Board's CPU, adding significant overhead on the inference time.

- Models aiming to run inference on the Edge TPU, must have a total size less than the Edge TPU's Cache (<8 MB,) after being converted to TFLite and compiled for the Edge TPU.

3.6.3 Edge Optimization Techniques

Apart from creating efficient devices, there are many techniques and conversions that can help reduce a model's size and latency. Below we present some of the most common ones.

3.6.3.1 Pruning

Neural Network pruning is a process that aims to reduce the size of a model, while minimizing the loss in performance and accuracy. It is a method of model compression that involves the removal of weights from a pretrained model, ultimately leading to networks smaller in terms of size, that require less training time and offer faster inference.

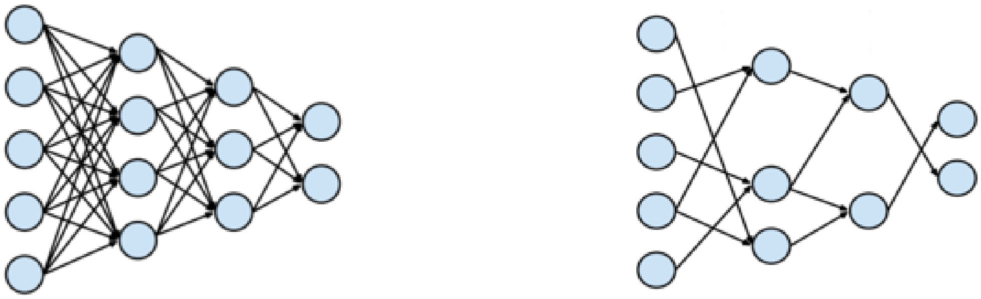


Figure 22. Visualization of the effects of pruning

The main idea behind pruning, is that while all neurons in an ANN are connected with every other neuron in adjacent layers through synapses, every neuron or synapse contributes differently in the final outcome. By ranking the contribution of the above, we can reduce the total size of the network by removing the lowest scoring neurons and synapses, while retaining a healthy size – accuracy balance.

Pruning techniques fall into two major categories, structured and unstructured. Unstructured pruning commonly involves directly removing parameters in an ANN, by setting their weighs to zero. This is a fine-grain approach that allows the removal of very specific parameters, even within convolution kernels. The disadvantage is that networks pruned this way contain many weights with zero value, and these so called “sparse” networks do not offer significant performance increase on most hardware architectures. On the other hand, structured pruning techniques aim to remove whole structures from the network, such as feature maps. This approach has a stronger impact on the resulting network, but should always be used with caution, as it can sometimes remove whole layers, leading to unconnected synapses.

Last but not least, before pruning there are various pruning criteria in order to decide the relative importance of the parameters. The weight magnitude criterion, that removes weights with the smallest absolute value, is one of the most commonly used and efficient of the aforementioned criteria used, even though it is not easy to implement in a structured way. Gradient magnitude pruning follows the same principle, but this time removing the parameters with the smallest gradient.

3.6.3.2 Quantization

Briefly, quantization is a process of converting weights and biases into a lower precision format in order to reduce the total storage size of a model. The parameters of an ANN are more often than not in a standard 32-bit floating-point arithmetic format. By converting these parameters into a lower precision format, like 16-bit floating point or 8-bit integers, we can effectively lower the storage size required for the model, reduce memory consumption and reduce latency during inference.

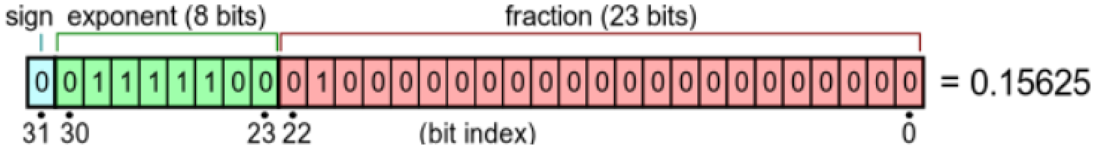


Figure 23. 32 Float format

For example, a single precision, or 32-bit floating point number can be represented as shown in Figure 23 where the actual value can be calculated in the decimal format with the following formula:

$$(-1)^{sign} \times (1 + fraction) \times 2^{exponent-bias} \qquad (10)$$

During quantization, for example 8-bit quantization, we can approximate this floating-point binary number into an 8-bit integer by using the follow equation:

$$int8_{value} = clip(round\left(\frac{real_{value}}{scale}\right) \qquad (11)$$

Where the real value can be calculated as:

$$real_{value} = \left(int8_{value} - zero_{point}\right) \times scale \qquad (12)$$

The result is an 8-bit integer in the range of [-128,127]. By lowering the precision, a model's size can be reduced up to four times. This performance increase comes with the danger of accuracy loss.

There are two commonly used kinds of quantization, post-training quantization and quantization-aware training. With post-training quantization, as the name implies, the neural network is trained with 32-bit floating point parameters and then quantized. Though this approach is easier, the parameters after quantization are frozen and there is no way to improve accumulative errors caused to lowering the precision. On the other hand, quantization-aware training tries to compensate for the quantization-related errors by utilizing the quantized weights during forward propagation.

3.6.3.3 Weight Clustering

Weight Clustering, or weight sharing, is an optimization technique that aims to reduce the total memory size of a model by reducing the number of unique weights in it. The first step in weight clustering is running a clustering algorithm over the weights of a specific layer, in order to obtain the desired number of centroids. Similar weights are replaced by the centroid's index they correspond to. The result is that instead of the weights, we can now store just an index table and a set of indices.

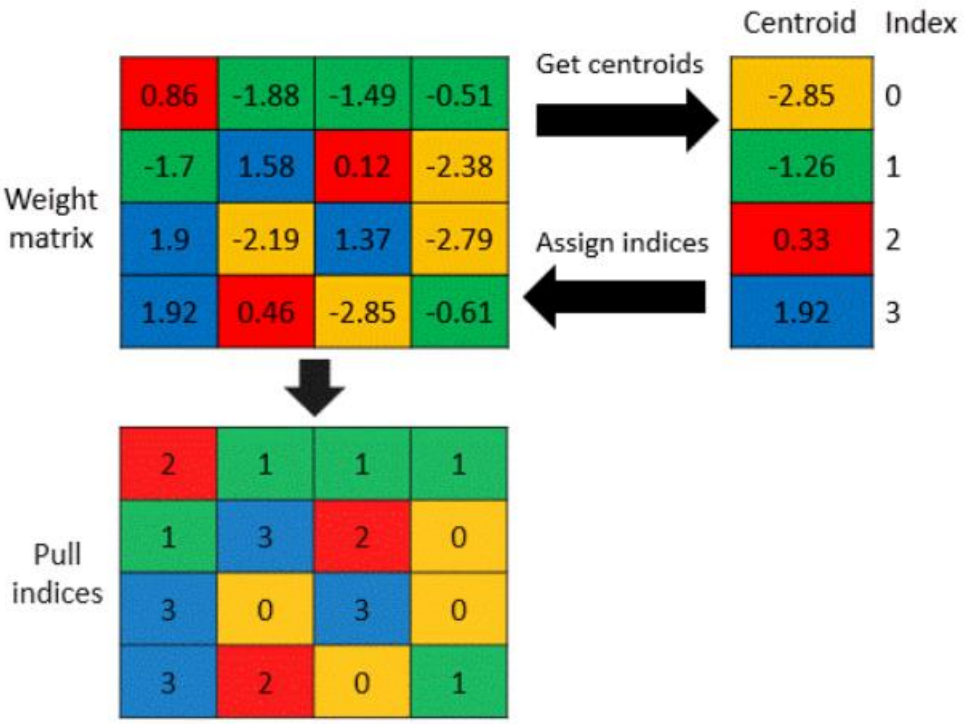


Figure 24. Weight clustering visualization

Weight clustering has an immediate advantage in reducing model storage and transfer size across serialization formats, as a model with shared parameters has a much higher compression rate than one without

4. Convolutional Neural Networks for Edge Applications

4.1 MobileNetV2

The MobileNet family of neural networks, is a well-established series of convolutional neural networks developed by Google, with the aim of improving performance and reducing inference time in mobile applications. MobileNet V1 introduced the concept of depthwise separable convolutions. Mobilenet V2 superseded V1, still utilizing the same type of depthwise separable convolutions but also introducing a new module called Inverted Residuals and Linear Bottlenecks.

For our use case we selected the MobileNet V2 CNN that was released in 2018. Even though by this time, MobileNet V3 has already succeeded it, according to Google Coral’s official benchmarks, MobileNet V2 seems to outperform its successor in terms of inference time. This difference in performance derives primarily from two factors:

1. MobileNet V3 makes use of the hard-swish activation function, which is not supported from the Edge TPU processor.

2. Like many of the state-of-the-art models, MobileNetV3 utilizes channel attention mechanisms, specifically squeeze-excitation modules. During our experimentation we concluded that these modules perform poorly on the Edge TPU processor in terms of runtime.

4.1.1 Inverted Residuals

The term residuals describe skip connections between the start and end of convolutional blocks. By using these skip connections, the CNN is able to access earlier activations that have not been modified by the convolutional block, ultimately improving performance the deeper the network gets.

Usual residual blocks followed a wide-narrow-wide approach, utilizing different kernel sizes in order to squeeze or expand the parameters in the channel dimension.

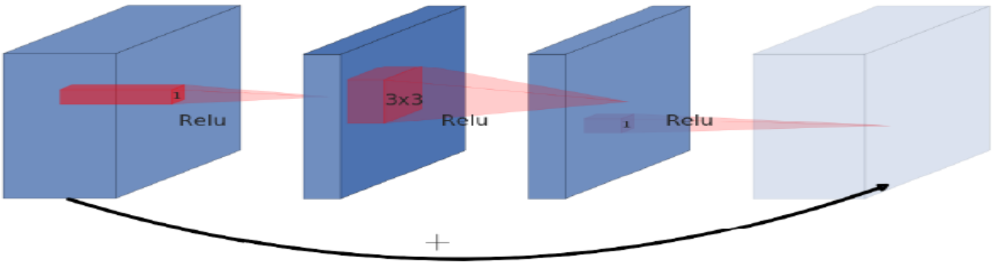


Figure 25. Conventional residual Block

MobileNetV2 introduced residual blocks following a narrow-wide-narrow approach, which, in contrast to the previous approach, carry a reduced number of parameters.

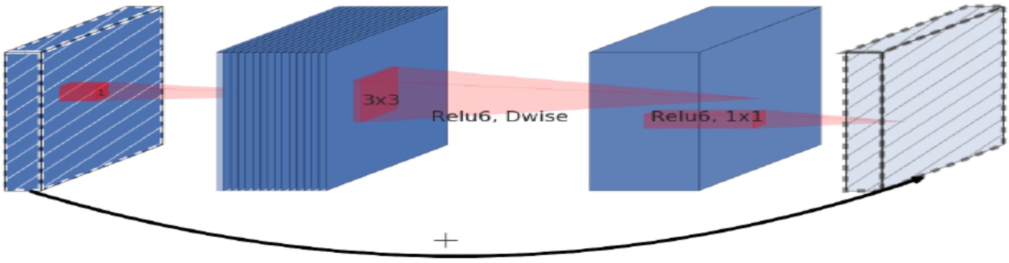


Figure 26. MobileNetV2 residual block

4.1.2 Linear Bottlenecks

Inverted residuals introduced skip connections after squeezed layers instead of expanded. This fact, in combination with the commonly used ReLU activation function, which discards values lower than zero, had a negative impact in the performance of the network. For this reason, the researchers introduced linear bottlenecks, where they essentially discard the activation function after the last convolutional layer of a block.

4.1.3 Architecture

Using the modules we mentioned above, MobileNetV2's convolutional blocks can be seen in Figure 27.

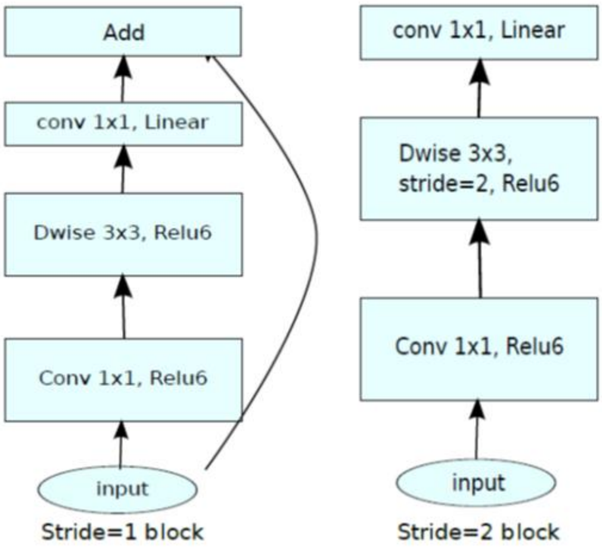
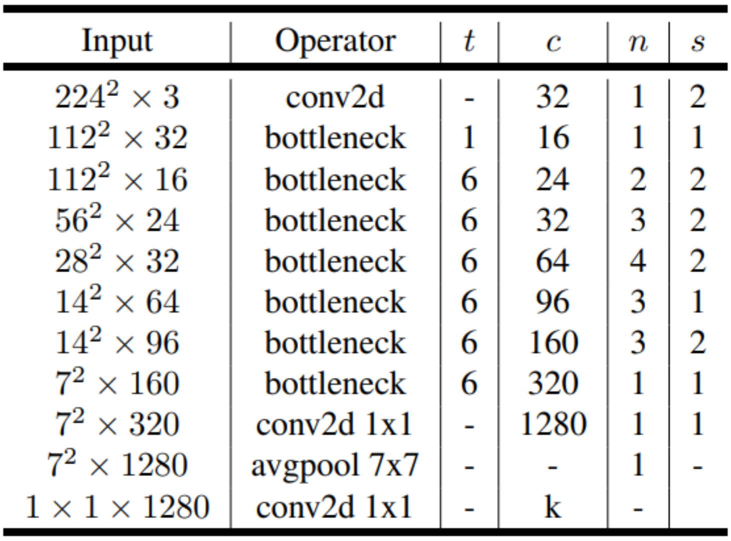


Figure 27. MobileNetV2 bottlenecks

The whole architecture can be seen in Figure 28, where t is the expansion factor, c the number of output channels, n is the repeating number and s the stride. Convolutional layers utilize a 3x3 kernel. The width of the network can also be modified by specifying a width multiplier.



Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Figure 28. MobileNetV2 architecture

MobileNetV2 with a width multiplier set to 1 and input size 224x224, requires a total of 300 million multiply-adds and has 3.4 million parameters. The network achieved a top-1 accuracy of 72% on the ImageNet classification task.

4.2 GhostNet

GhostNet: More Features from Cheap Operations was introduced in CVPR 2020 and was one of the most notable breakthroughs that year. The idea behind this paper is that many of the feature maps generated from a convolutional layer are redundant or show a high degree of similarity. With this in mind, the researchers proposed an alternative method to calculate a percentage of the total feature maps, which requires less computational effort than the standard convolution. Using these methods and the MobileNetV3 CNN as a backbone, they presented the GhostNet CNN.

4.2.1 Feature Map Redundancy and the GhostNet Module

Taking a closer look at the feature maps generated by a convolutional layer in Figure 29, researchers noticed that there many similar copies of unique intrinsic feature maps generated through the computationally expensive convolution. They called these feature maps “Ghost Feature Maps” and they proposed a way to generate them using a cheap operation, the Ghost Module.

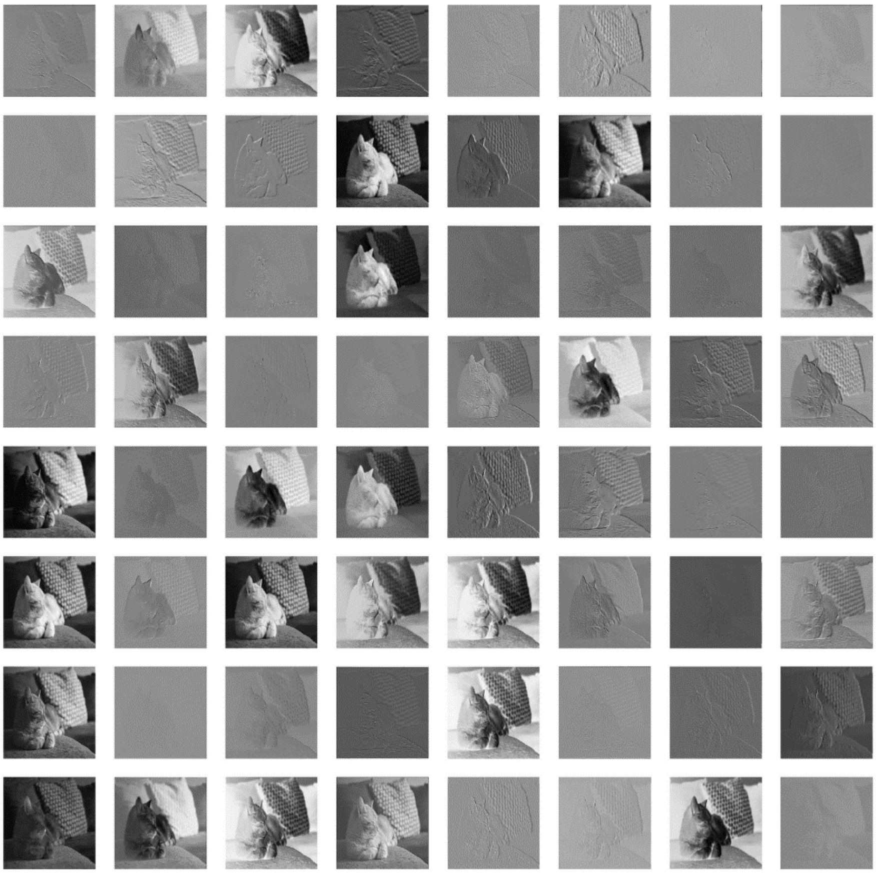


Figure 29. GhostNet feature maps

The main goal of the Ghost Module is to replace the standard convolutional layer and reduce the FLOPS required. Assuming the input and output tensors of a convolutional layer, with C and C' channels respectively, the Ghost Module works as follows:

1. Compute a percentage $x\%$ of the desired output's feature maps, xC', through standard convolution. Pass the output through a batch normalization layer and a ReLU activation function.
2. Using the output of step one as input, compute the rest of the feature maps, $(1-x)C'$, through a depthwise convolution. Again, pass the output through a batch normalization layer and a ReLU activation function.
3. Stack the results of the two steps along the channel axis.

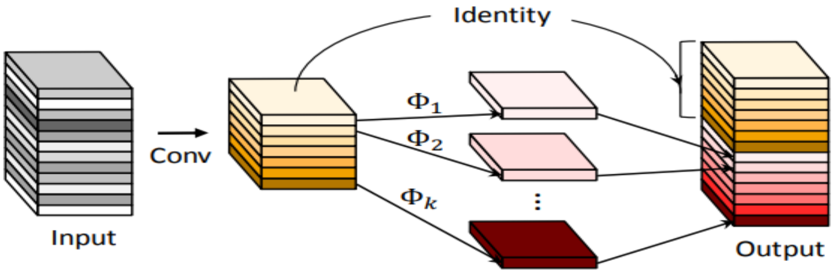


Figure 30. Ghost module

4.2.2 Architecture

Utilizing the Ghost Module, the researchers proposed a new backbone architecture called GhostNet. This architecture is essentially a "ghosted" MobileNetV3, where the bottleneck is replaced by the Ghost Bottleneck, as seen in Figure 31.

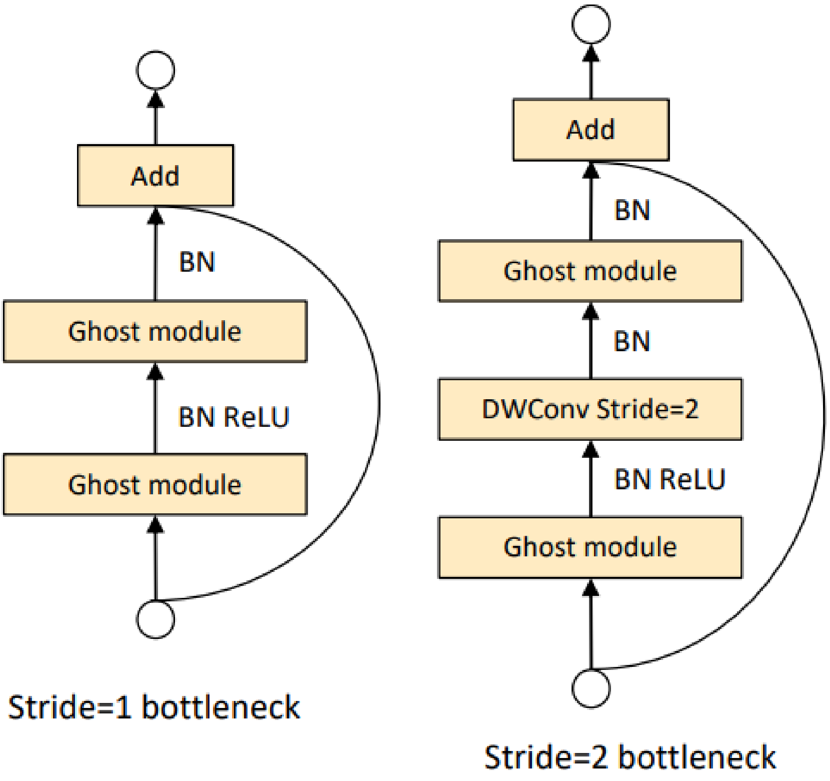
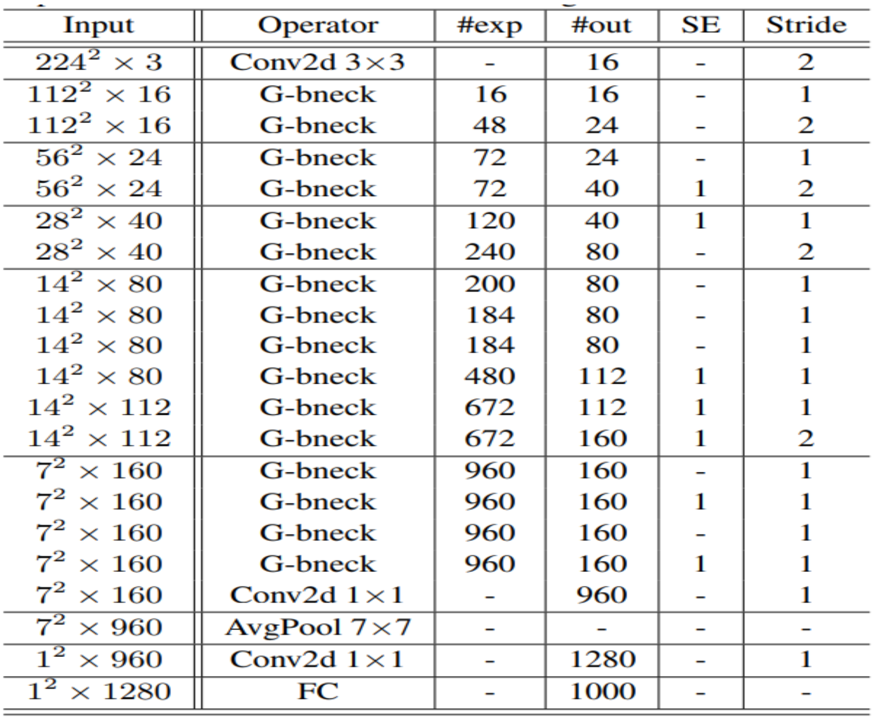


Figure 31. Ghost Bottlenecks

GhostNet is comprised from a stem convolution layer, which was not replaced, several grouped in stages Ghost Bottlenecks following an incrementing number of channels and a classifier head. In all the Ghost Bottlenecks a stride of 1 was applied, except for the last bottleneck where the stride 2 design was used. For some residual connections in the Ghost bottlenecks the developers also used Squeeze-Excitation (SE) blocks to provide channel attention, thus improving accuracy with a small computational overhead. The whole architecture can be better inspected in Figure 32.



Input	Operator	#exp	#out	SE	Stride
$224^2 \times 3$	Conv2d 3×3	-	16	-	2
$112^2 \times 16$	G-bneck	16	16	-	1
$112^2 \times 16$	G-bneck	48	24	-	2
$56^2 \times 24$	G-bneck	72	24	-	1
$56^2 \times 24$	G-bneck	72	40	1	2
$28^2 \times 40$	G-bneck	120	40	1	1
$28^2 \times 40$	G-bneck	240	80	-	2
$14^2 \times 80$	G-bneck	200	80	-	1
$14^2 \times 80$	G-bneck	184	80	-	1
$14^2 \times 80$	G-bneck	184	80	-	1
$14^2 \times 80$	G-bneck	480	112	1	1
$14^2 \times 112$	G-bneck	672	112	1	1
$14^2 \times 112$	G-bneck	672	160	1	2
$7^2 \times 160$	G-bneck	960	160	-	1
$7^2 \times 160$	G-bneck	960	160	1	1
$7^2 \times 160$	G-bneck	960	160	-	1
$7^2 \times 160$	G-bneck	960	160	1	1
$7^2 \times 160$	Conv2d 1×1	-	960	-	1
$7^2 \times 960$	AvgPool 7×7	-	-	-	-
$1^2 \times 960$	Conv2d 1×1	-	1280	-	1
$1^2 \times 1280$	FC	-	1000	-	-

Figure 32. GhostNet architecture

4.2.3 GhostNetEdgeTPU

In addition to the original GhostNet, we also introduce a modified variation of it, called GhostNetEdgeTPU, which is more suitable for inference on edge devices. Specifically, retaining the original architecture, we remove the channel attention modules, squeeze-excitation modules, and we replace the ReLU activation function with ReLU6.

4.3 MobileNetEdgeTPU V2

Google has developed MobileNetEdgeTPUV2 as a set of models for carrying out efficient on-device inference. This particular architecture is the outcome of using Neural Architecture Search (NAS), which is a method for automatically designing neural network architectures that are efficient. One of the important aspects of using NAS involves creating a "search space" containing various potential modules that can be evaluated based on desired metrics, ultimately leading to the final architecture.

4.3.1 Fused Inverted Bottlenecks

One widely-used building block in neural networks for various on-device vision tasks is the Inverted Bottleneck (IBN), as explained in chapter 4.1. The IBN block has several variants, each with different tradeoffs, and is built using regular convolution and depthwise convolution layers. While IBNs with depthwise convolution have been conventionally used in mobile vision models due to their low computational complexity, fused-IBNs, wherein depthwise convolution is replaced by a regular convolution, have been shown to improve the accuracy and latency of image classification models on TPU.

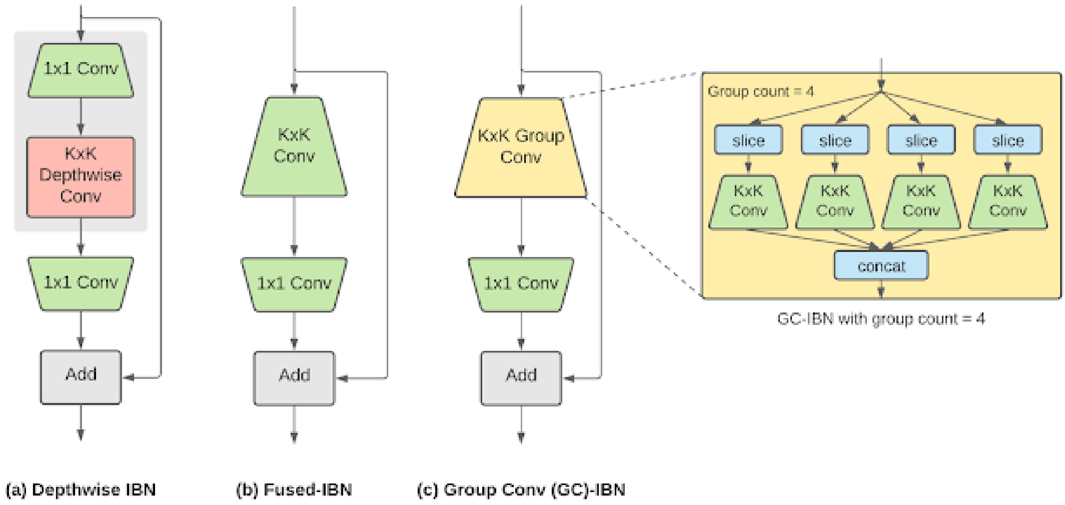


Figure 33. Inverted Bottleneck variants

However, fused-IBNs, in contrast to the depthwise-IBN, come with high computational and memory requirements, especially for convolutional layers at the latter stages of a neural network. In order to overcome this obstacle, the researchers introduced IBNs that utilize grouped convolutions, further expanding the search space and improving model flexibility. Grouped convolutions essentially slice the features maps into smaller subsets before performing the convolution. This results in significantly reduced computational cost. These blocks, also called Group Convolution-based IBNs (GC-IBNs), provide a way to balance the trade-off between model quality and latency.

4.3.2 Architecture

Which IBN variant to use at which stage of a deep neural network depends on the latency on the target hardware and the performance of the resulting neural network on the given task. We construct a search space that includes all of these different IBN variants and use NAS to discover neural networks for the image classification task that optimize the classification accuracy at a desired latency on TPU. The resulting MobileNetEdgeTPUV2 model family improves the accuracy at a given latency (or latency at a desired accuracy) compared to the existing on-device models when run on the TPU. MobileNetEdgeTPUV2 also outperforms their predecessor, MobileNetEdgeTPU, the image classification models designed for the previous generation of the TPU.

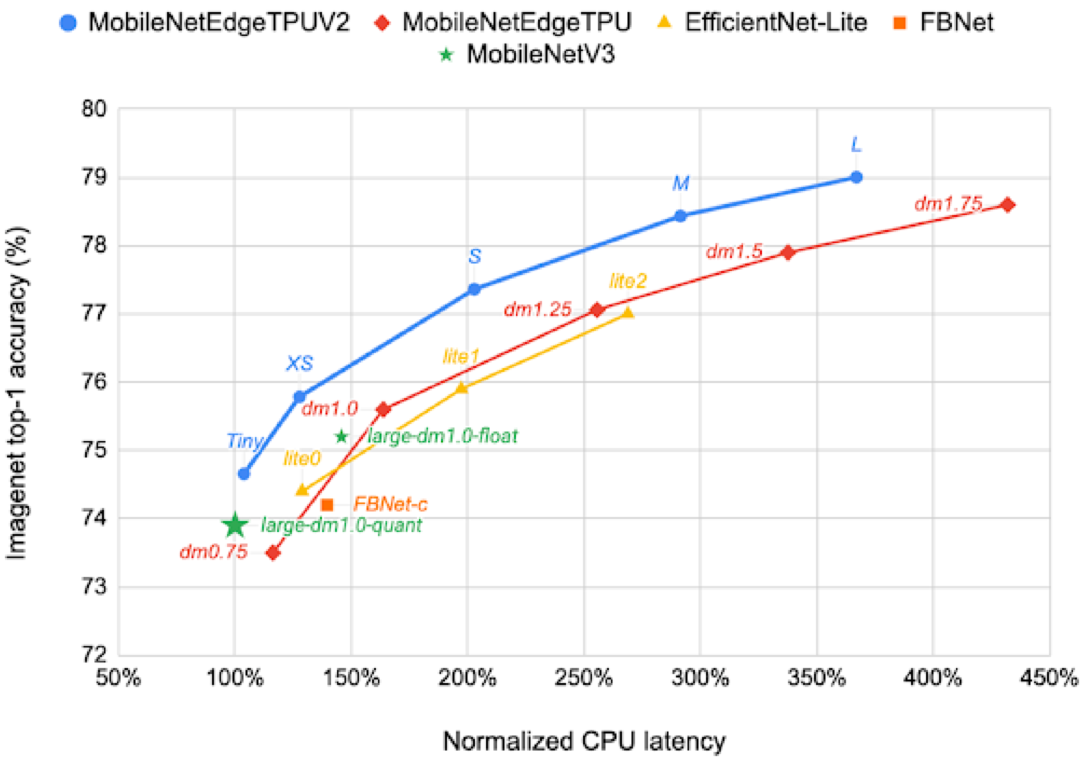


Figure 34. MobileNetEdgeTPUV2 performance comparison on Pixel 6 CPU

MobileNetEdgeTPUV2 models are built using blocks that also improve the latency/accuracy tradeoff on other compute elements in the Google Tensor SoC, such as the CPU. Unlike accelerators such as the TPU, CPUs show a stronger correlation between the number of multiply-and-accumulate operations in the neural network and latency. GC-IBNs tend to have fewer multiply-and-accumulate operations than fused-IBNs, which leads MobileNetEdgeTPUV2 to outperform other models even on Pixel 6 CPU.

4.4 EfficientNet-Lite

The EfficientNet family of CNNs is considered state-of-the-art since its introduction by Google AI in 2019. The main goal of the EfficientNet is to provide a method of scaling model architectures in a principled and effective manner, in contrast to older techniques that required manual oversight. To achieve this, the authors introduced compound coefficient scaling and based on it, seven different model variants - EfficientNet B0-B7 - that obtained state-of-the-art performance on the ImageNet challenge.

4.4.1 Compound Coefficient Scaling

In search for an efficient method of compound scaling, the researchers studied the impact of different scaling techniques on model performance. Ultimately, they reached the conclusion that even though scaling single dimensions can help improve performance, balancing the scaling across all three dimensions-width, depth and resolution- yielded better performance overall.

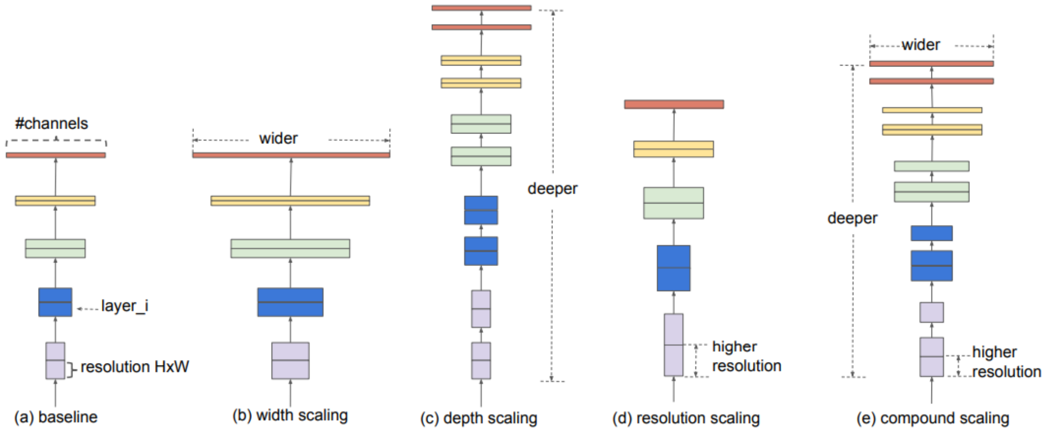


Figure 35. Single scaling options vs. Compound Scaling

In order to keep the scaling balanced, a constant ratio φ representing the increase in computational resource to the network, also called compound coefficient, is used, in a way that:

$$depth = a^{\varphi}\ , \quad width = \beta^{\varphi},\ resolution = \gamma^{\varphi}$$

So that:

$$\alpha * \beta^2 * \gamma^2 \approx 2 \tag{13}$$

Where the variables $\alpha, \beta, \gamma\ \geq 1$ are determined using a grid search algorithm. The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image.

4.4.2 EfficientNet-Lite

Following the success of the EfficientNet classification models, Google released the EfficieNet-Lite family of CNNs, a variation of the original models deisgned for performance on mobile CPUs, GPUs and EdgeTPUs.

4.4.2.1 Architecture

The base EfficientNet-B0 network is based on the inverted bottleneck residual blocks of MobileNetV2, as described previously.

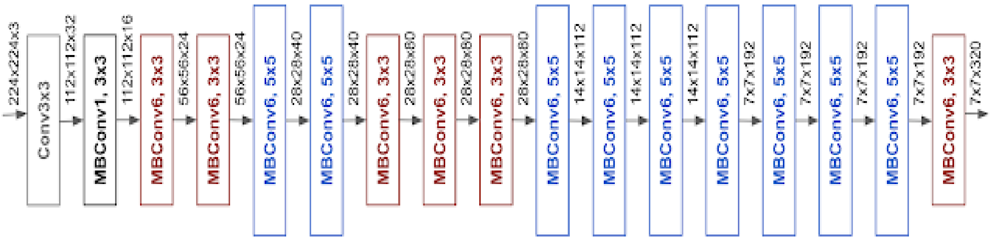


Figure 36. EfficientNetB0 architecture

EfficientNet-Lite, builds further on this model, making it more suitable for mobile devices by introducing ReLU6 activation functions and removing the squeeze-excitation blocks.

4.5 Ensemble Models

In addition to the rest of the CNNs presented on this chapter, we also introduce two more models base on Ensemble Learning. Since ensemble models comprise of a number of smaller in size models, they tend to be bigger in terms of parameters, but also more efficient. For these reasons, they provide a good measure for the size-accuracy trade-off, but also offer insight on how the width and depth of a model affect the inference time on the EdgeTPU.

4.5.1 Ensemble Model 1

The first ensemble model follows a more traditional architecture, by utilizing a combination of the previously described CNN architectures.

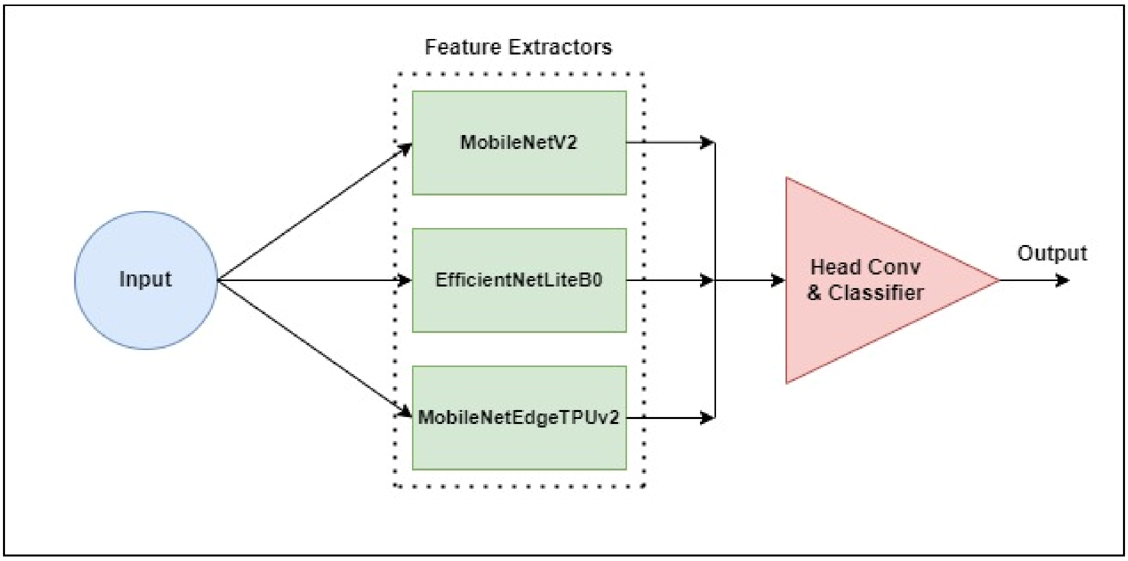


Figure 37. Building blocks of ensemble model 1

In this approach, three of the previously described models are used as feature vectors in order to extract critical information from the input images. Their results are later combined and passed through a final convolution and a classifier.

Specifically, the three models used are the MobileNetv2, the EfficientNetLiteB0 and the MobileNetEdgeTPUv2. These models are already individually pretrained on the Kvasir-v2 dataset.

For the purpose of utilizing them as feature vectors, the final convolution and the classifier of each model are being stripped and the remaining layers are frozen during the final training procedure, where we aim only to train the combined convolution and the final classifier.

4.5.2 Ensemble Model 2

The second model proposed comes after having trained and evaluated all the previous models. Its purpose is to address weaknesses, which were identified and found to be common to all previous models. Further insight into the logic behind this model is provided in Chapter 6.

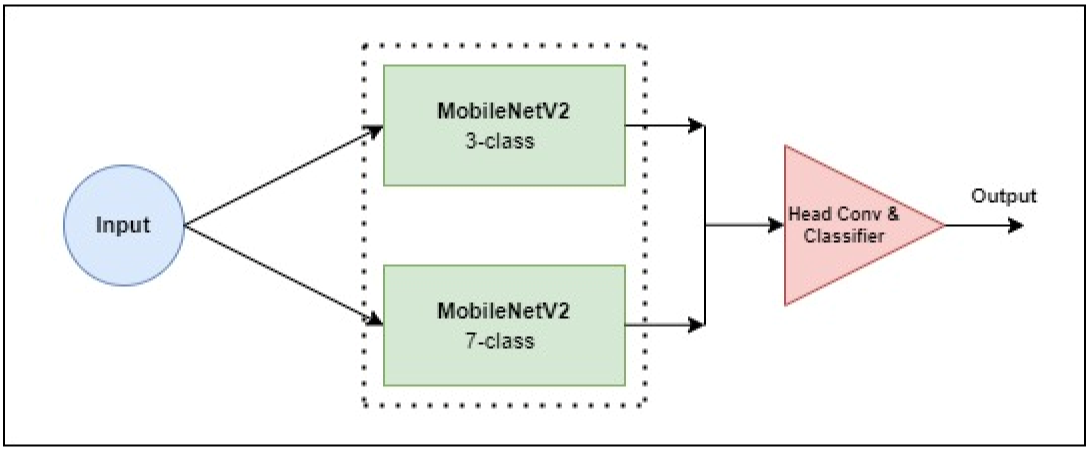


Figure 38. Building blocks of ensemble model 2

Its architecture is similar to the previous ensemble model. It consists of two MobileNetV2 models that are utilized as feature vectors. They share a common input and after stripping the final layers of the base models, their outputs are combined at the final stage and pass through a convolutional layer and the classifier.

It is important to note that in contrast to the previous ensemble model, due to several differences in this approach that will be further explained in Chapter 5, these models are trained from scratch.

5. Methodology

5.1 Data Pre-processing

The image pre-processing pipeline consists of a total of four steps:

1. Split images into training/test/validation
2. Perform Modular Adaptive Pre-processing for GI tract images (MAPGI)
3. Resize and rescale the images appropriately for training
4. Increase the total number of images through data augmentation

These steps are explained in detail on the following sections.

5.1.1 Split Images

5.1.1.1 Default Process

The dataset includes 8.000 images in total. For training purposes, these images have been split into training – validation – test subsets with a ratio of 0.765 – 0.135 – 0.1 respectively. This results in a training set of 6.120 images, 765 of each class, a validation set of 1.080 images, or 135 of each class and a test set of 800 images, 100 from each class respectively.

5.1.1.2 Special Case

Ensemble model 2 is considered a special case and requires a different splitting method. The two MobileNetV2 networks that comprise the ensemble model are trained on two different subsets of the Kvasir v2 dataset. We refer to these datasets as **3-class dataset** and **7-class dataset**.

The **3-class dataset** consists of a total of three classes, two originally found in the Kvsair v2 dataset, the Esophagitis and the Normal Z-line classes, and a new class called Other, which consists of random samples from the six remaining classes. The **7-class dataset** follows the same logic, with the difference that it consists of a total of seven classes. These are six classes of the original dataset, specifically all classes except the Esophagitis and Normal Z-line classes. The latter two are randomly sampled and combined into a new class called Other. Each class from both subsets has a total of 1000 images that are splatted into train-validation-test with ratio 0.765 – 0.135 – 0.1 respectively.

5.1.2 MAPGI Framework

The Modular Adaptive Preprocessing for Gastrointestinal Tract images framework, or MAPGI, was introduced by T.Cogan and M.Cogan [13] as a way to improve the performance of CNNs on the classification task of the Kvasir v2 dataset. Within the framework, images are represented in the YUV color space, instead of the common RGB, because the Y component solely encodes image luminance, and as such it can be treated as a grayscale image itself, making the framework robust against different color spaces.

The framework consists of a total of 5 steps, which are described in detail in the following sections.

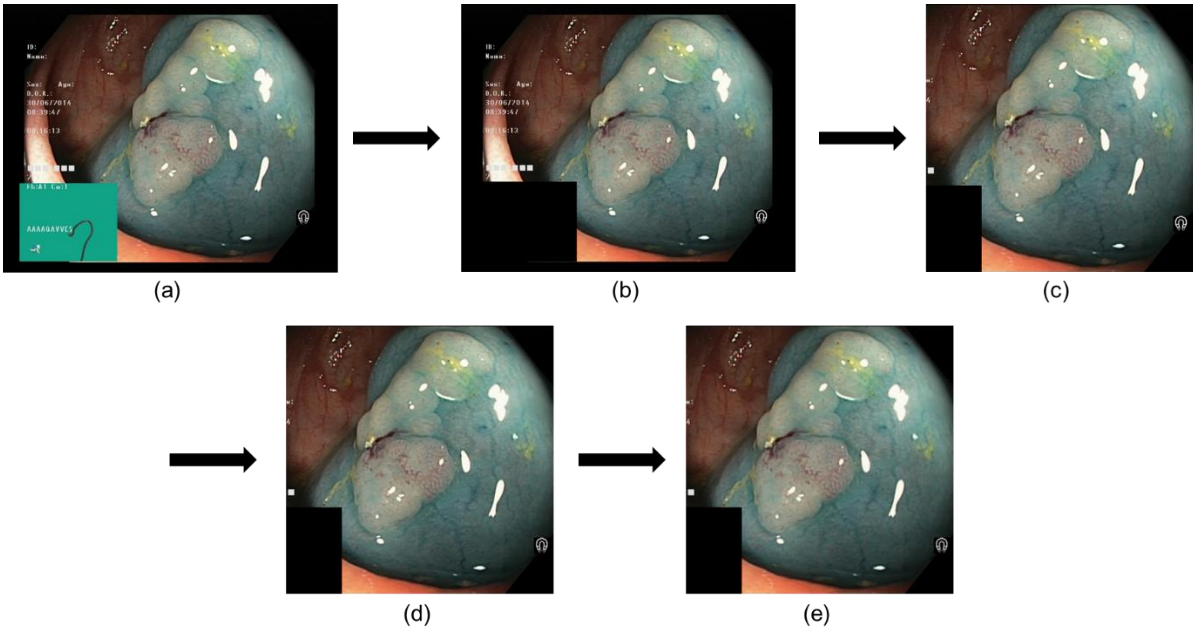

(a) (b) (c) (d) (e)

Figure 39 MAPGI framework's steps: (a) Masking, (b) Crop, (c) MAVGA, (d) Lowpass Filter, (e) Resize & Rescale

5.1.2.1 Masking

As mentioned before, many images contain a small box on the left bottom corner, depicting the position of the endoscope. This information is not always available and can mislead ANNs into learning from information that will not be available in the future, negatively impacting their robustness. For this reason, at the first stage of the pre-processing framework, an image processing algorithm has been used in order to mask the aforementioned box. Because the position and the colour of the box are constants, this algorithm detects the box by measuring pixel intensity on all three channels and then proceeds to mask the pixels detected.

5.1.2.2 Intelligent Cropping

On the second step, we appropriately crop the images in order to reduce areas of the image that contain no valuable information for the ANNs, like black edges etc. This procedure is performed by checking the mean pixel intensity for row and columns, starting from all the four borders of the image. Every row or column that has a mean pixel intensity lower than a user specified threshold, is cropped. The cropping continues until a row of pixels is met, which has a pixel intensity higher than the threshold specified.

5.1.2.3 MAVGA Module

The MAVGA Module, or Mean-Approximated Gamma Value Adjustment Module, is a function for performing contrast enhancement via gamma correction, that was proposed by the same authors, instead of most commonly used methods, such as Contrast-Limited Adaptive Histogram Equalization.

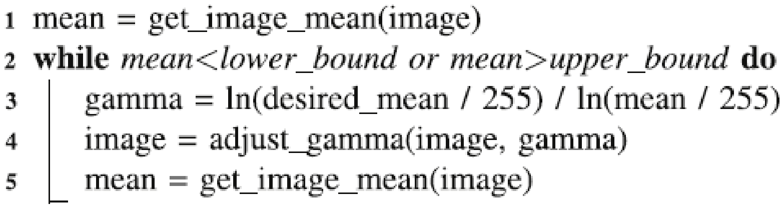
The MAVGA Module uses recursive method in order to perform gamma correction. At first, we compare the mean pixel value of an image to the desired mean pixel value, in our case 90 ± 1. After this, the algorithm estimated a gamma value needed in order to correct the image's brightness, and then applies a gamma correction. This procedure is repeated until the mean pixel value is in the desired range. Specifically, let b be the desired mean pixel value then we want:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{I_i}{255}\right)^{\lambda} = b \tag{15}$$

Where I_i represents pixel values and λ is the desired gamma correction coefficient. Replacing the pixels by the average pixel value of the image, a, and solving for λ:

$$\lambda = \frac{\ln b - \ln 255}{\ln a - \ln 255} \tag{16}$$

The process works recursively as follows:



```
mean = get_image_mean(image)
while mean<lower_bound or mean>upper_bound do
    gamma = ln(desired_mean / 255) / ln(mean / 255)
    image = adjust_gamma(image, gamma)
    mean = get_image_mean(image)
```

5.1.2.4 Lowpass filter

At the last step of the MAPGI framework, we use a simple low-pass filter in order to reduce out-of-band noise. This filtering is only applied to the luminance channel. The filter is performed by convolving a kernel {[0.1, 0.1, 0.1], [0.1, 1, 0.1], [0.1, 0.1, 0.1]}/1.8 kernel through the entire image.

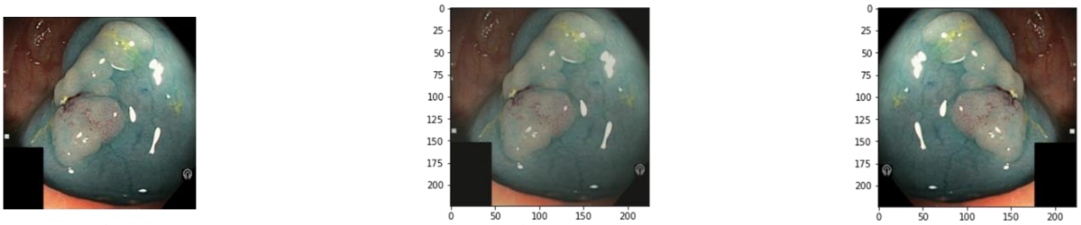
5.1.3 Resize & Rescale

As mentioned before, the images are in the RGB format with image resolutions that vary from 720x576 to 1920x1072. Following the MAPGI framework, the images are resized down to 224x224, due to the fact that this is the native input size for many of the CNNs utilized. Another vital step is the rescaling of the images. In the RGB format every pixel is in the range [0,255] but before using them for training, we rescale them so that every pixel has values in the range [0,1], or [-1,1] in cases concerning the EfficientNet family of CNNs.

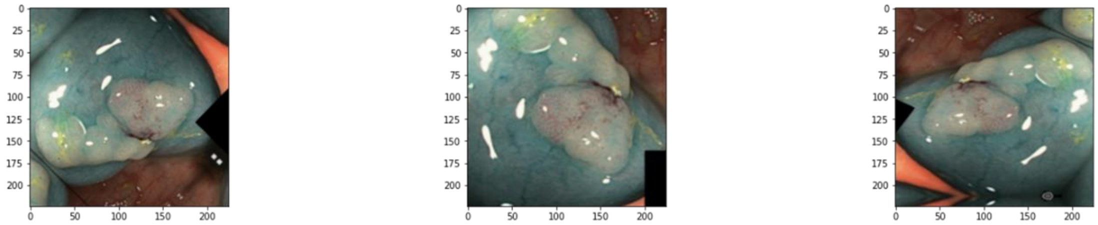
5.1.4 Image Augmentation

In order to increase the effective size of the dataset and counter overfitting, we used a set of image augmentation transformations that seemed appropriated for our use case. These transformations are the following:

- Random Flip, Horizontal and Vertical
- Random Rotation with rotation factor 0.45
- Random Contrast with contrast factor 0.30
- Random Zoom with zoom factor 0.30



Original Image *Random Contrast* *Random Flip*



Random Rotation *Random Zoom* *All Transformations*

Figure 40. Image Augmentation example

5.2 Training Configuration

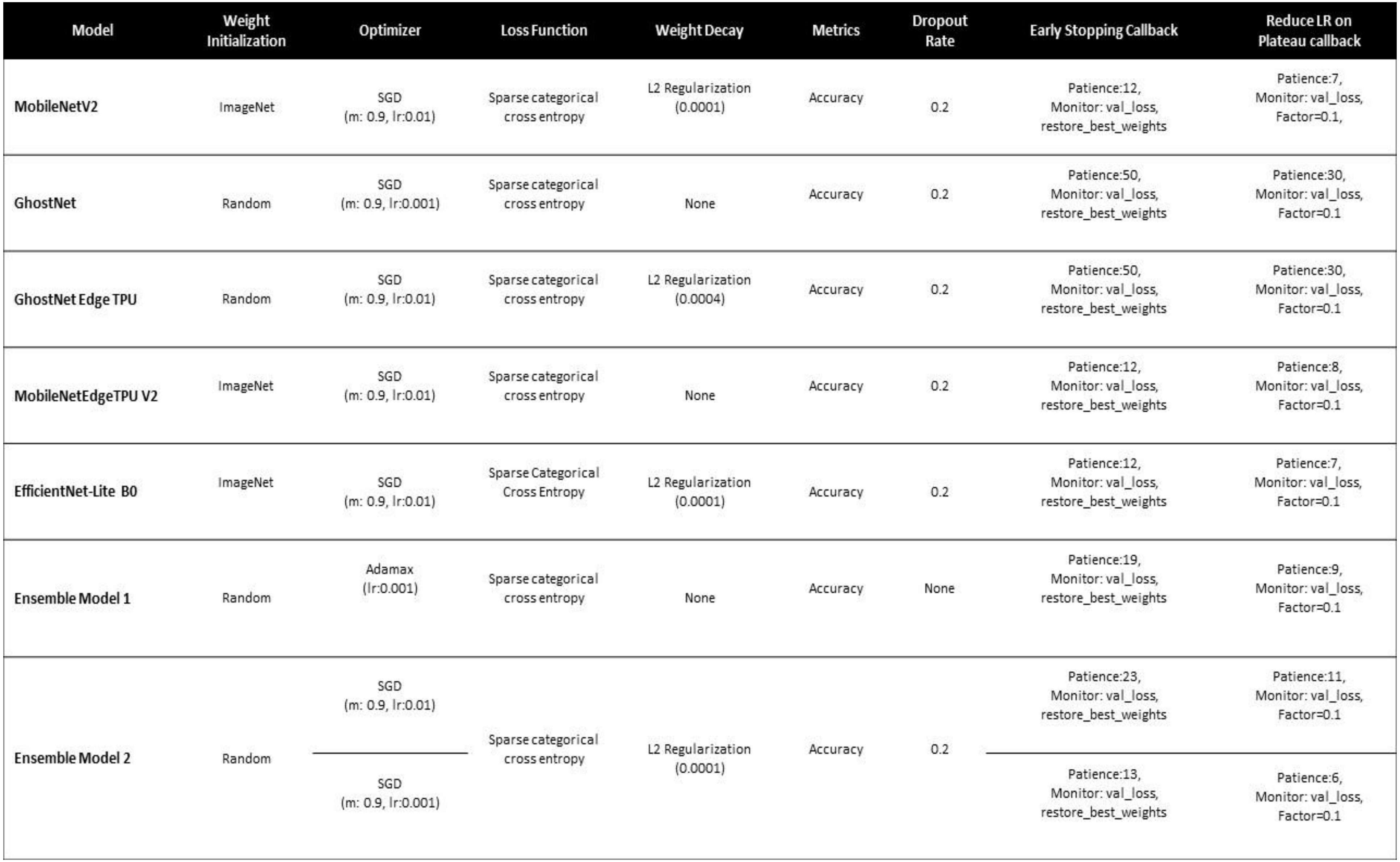
5.2.1 Training Setup

All data and image processing scripts have been implemented using the Python v.3.9.0 programming language. Specifically for the development of the MAPGI framework, the OpenCV library has been used. All model architectures have been implemented using the Keras and TensorFlow 2.7.0.

CNN models have been trained and evaluated on a desktop consisting of an AMD Ryzen 5 3600X 6-core processor, 16 GB DDR4 RAM and a RTX 3060Ti NVIDIA GPU.

5.2.2 Training Parameters

Below, we briefly present the training parameters as set before training:



Model	Weight Initialization	Optimizer	Loss Function	Weight Decay	Metrics	Dropout Rate	Early Stopping Callback	Reduce LR on Plateau callback
MobileNetV2	ImageNet	SGD (m: 0.9, lr:0.01)	Sparse categorical cross entropy	L2 Regularization (0.0001)	Accuracy	0.2	Patience:12, Monitor: val_loss, restore_best_weights	Patience:7, Monitor: val_loss, Factor=0.1,
GhostNet	Random	SGD (m: 0.9, lr:0.001)	Sparse categorical cross entropy	None	Accuracy	0.2	Patience:50, Monitor: val_loss, restore_best_weights	Patience:30, Monitor: val_loss, Factor=0.1
GhostNet Edge TPU	Random	SGD (m: 0.9, lr:0.01)	Sparse categorical cross entropy	L2 Regularization (0.0004)	Accuracy	0.2	Patience:50, Monitor: val_loss, restore_best_weights	Patience:30, Monitor: val_loss, Factor=0.1
MobileNetEdgeTPU V2	ImageNet	SGD (m: 0.9, lr:0.01)	Sparse categorical cross entropy	None	Accuracy	0.2	Patience:12, Monitor: val_loss, restore_best_weights	Patience:8, Monitor: val_loss, Factor=0.1
EfficientNet-Lite B0	ImageNet	SGD (m: 0.9, lr:0.01)	Sparse Categorical Cross Entropy	L2 Regularization (0.0001)	Accuracy	0.2	Patience:12, Monitor: val_loss, restore_best_weights	Patience:7, Monitor: val_loss, Factor=0.1
Ensemble Model 1	Random	Adamax (lr:0.001)	Sparse categorical cross entropy	None	Accuracy	None	Patience:19, Monitor: val_loss, restore_best_weights	Patience:9, Monitor: val_loss, Factor=0.1
Ensemble Model 2	Random	SGD (m: 0.9, lr:0.01)	Sparse categorical cross entropy	L2 Regularization (0.0001)	Accuracy	0.2	Patience:23, Monitor: val_loss, restore_best_weights	Patience:11, Monitor: val_loss, Factor=0.1
		SGD (m: 0.9, lr:0.001)					Patience:13, Monitor: val_loss, restore_best_weights	Patience:6, Monitor: val_loss, Factor=0.1

Table 3 Training Parameters

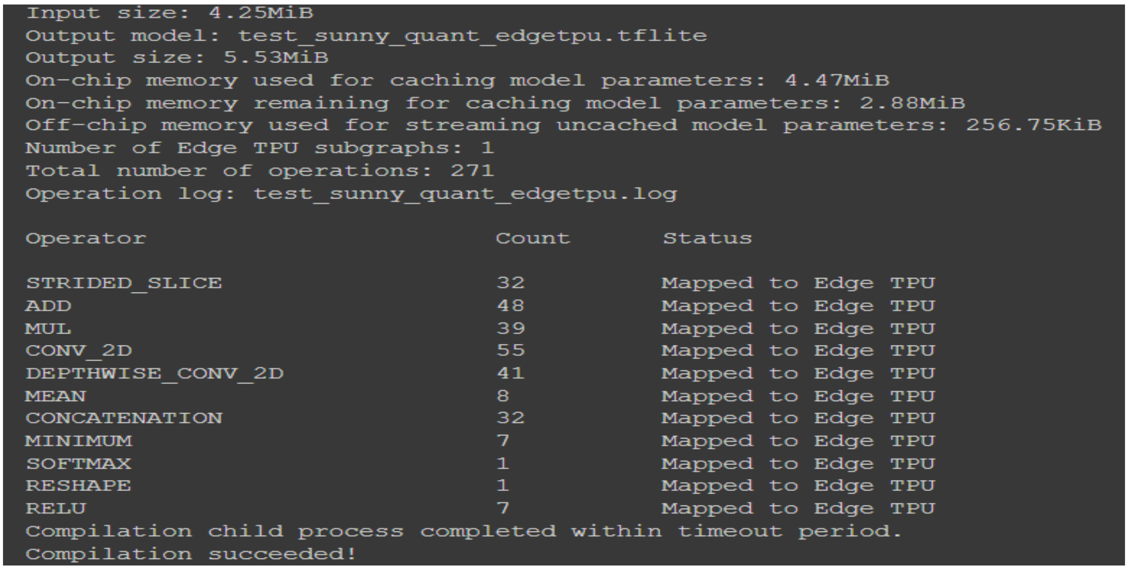
5.3 Post Training Optimization

5.3.1 Post Training Quantization

Following training, optimization techniques for edge inference and model compression take place for all CNNs. This includes the methods explained in Ch 2., like quantization and pruning. This process leads to models in TensorFlow Lite format, which are suitable for edge applications.

5.3.2 Compilation for the Edge TPU processor

In order to prepare the quantized models for inference on Google Coral Edge TPU, they have to be compiled. During this procedure, all model operations are mapped out and optimized for the edge accelerator.



```
Input size: 4.25MiB
Output model: test_sunny_quant_edgetpu.tflite
Output size: 5.53MiB
On-chip memory used for caching model parameters: 4.47MiB
On-chip memory remaining for caching model parameters: 2.88MiB
Off-chip memory used for streaming uncached model parameters: 256.75KiB
Number of Edge TPU subgraphs: 1
Total number of operations: 271
Operation log: test_sunny_quant_edgetpu.log

Operator                       Count      Status

STRIDED_SLICE                  32         Mapped to Edge TPU
ADD                            48         Mapped to Edge TPU
MUL                            39         Mapped to Edge TPU
CONV_2D                        55         Mapped to Edge TPU
DEPTHWISE_CONV_2D              41         Mapped to Edge TPU
MEAN                           8          Mapped to Edge TPU
CONCATENATION                  32         Mapped to Edge TPU
MINIMUM                        7          Mapped to Edge TPU
SOFTMAX                        1          Mapped to Edge TPU
RESHAPE                        1          Mapped to Edge TPU
RELU                           7          Mapped to Edge TPU
Compilation child process completed within timeout period.
Compilation succeeded!
```

Figure 41. Edge TPU compiler report

One of the main considerations of this process is the utilization of the Edge TPU’s Cache. If the converted model’s size is small enough and all operations are supported, the compiler will map all operations to the aforementioned cache, improving the inference time. Bigger size models, or models with not allowed operations, can still run on the processor, but will use the off-chip memory. In this case, the inference time suffers significantly

5.4 Inference

In order to conduct inference, we setup the Google Coral Dev Board mini. Then we proceed to download all the compiled models, the test images as well as a benchmark model. Google Coral's website offers a variety of already tested models with their benchmarks. In order to establish a more accurate baseline for our tests, we downloaded their official EfficientNet-EdgeTpu-S model and compared the inference time with their benchmarks. This is necessary in order to better identify the impact of the differences between the Google Coral Dev Board and its mini version. Even though the AI accelerator is exactly the same between these two versions, the mini version utilizes a USB connection between the edge TPU and the board's CPU, in contrast to the PCI-E connection of the original board. Additionally, the mini version has less powerful system-on-module (SoC) with a dual-core processor, that might affect the time required for processing images through the MAPGI network.

Utilizing all the above, we test the quantized models directly on the Board, in order to compare their inference time. We also calculate the average time the MAPGI pre-processing framework requires for the processing of one image on the Edge TPU.

5.5 Evaluation

5.5.1 Metrics

In order to evaluate the models, we use all the suggested metrics from the original Kvasir v2 paper. For this reason, we conduct inference using the test images that have been hidden from the models and collect the following basic metrics:

True Positive (TP): The number of correctly classified positive images.

True Negative (TN): The number of correctly classified negative images.

False Positive (FP): The number of falsely classified positive images.

False Negative (FN): The number of falsely classified negative images.

It is important to note that these basics metrics are extracted for each class separately, since the task at hand involves multi-class classification. For a single class, positive is considered a sample belonging to the specific class, while negative is considered a sample belonging to any other class.

In the next step we calculate the following advanced metrics for each class, based on the previous ones:

Accuracy is the percentage of correctly classified images.

$$Accuracy = \frac{TP + TN}{\#\ samples\ in\ total} \tag{17}$$

Precision, also called the positive predictive value, represents the ratio of correctly classified positive samples among all returned values.

$$Precision = \frac{TP}{\#\ all\ returned\ samples} = \frac{TP}{TP + FP} \tag{18}$$

Recall also known as sensitivity, probability of detection or true positive rate, represents the ratio of samples that are classified as positive among all positive samples.

$$Recall = \frac{TP}{\#\ all\ positives} = \frac{TP}{TP + FN} \tag{19}$$

Specificity, or true negative rate, shows the ratio of the negatives that are correctly classified as such.

$$Specificity = \frac{TN}{\#\ all\ negatives} = \frac{TN}{FP + TN} \tag{20}$$

Matthew Correlation Coefficient (MCC) takes into account true and false positives and negatives, and is an efficient metric for unbalanced classes.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \qquad (21)$$

F1 Score is a measure of test’s accuracy through the calculation of the harmonic mean of the precision and recall.

$$F1\ score = 2 \times \frac{precision \times recall}{precision + recall} = \frac{2TP}{2TP + FP + FN} \qquad (22)$$

After obtaining these for each individual class, we then calculate the weighted average to procure the final results for a model.

In addition to the above metrics, we also take into consideration the total inference time as measured on the Edge TPU, the size of a quantized model and the total count of its parameters. These added metrics are of crucial importance for edge applications.

5.5.2 Explainability

Saliency maps are not a metric but a group of visualization techniques used in the field of explainable artificial intelligence to help understand the features of an image that a convolutional neural network is using to make its predictions. In a CNN, each layer applies a set of filters to the input image, creating a feature map that highlights certain features or patterns in the image. Saliency maps use this information to create a heatmap that indicates the importance of different regions of the input image for the CNN's prediction. Saliency maps are typically generated by computing the gradient of the output of the CNN with respect to the input image, and then normalizing the gradients to highlight the most important regions of the image. This allows us to see which parts of the image the network is paying the most attention to when making its prediction, which can help provide insights into how the network is making its decision. By examining the saliency map, we can identify the most important features of an image for a given classification task. This can help us better understand how the neural network is making its predictions, which in turn can help us improve the network's performance, troubleshoot issues, and ensure that the network is making decisions in a way that aligns with our expectations.

In Chapter 6 we will use one of the simplest techniques called **Gradient-based Saliency** (Vanilla Saliency) in order to provide insight and better understand our models and input data. Vanilla Saliency works by computing the gradient of the output class score with respect to the input image pixels. The gradient values reflect how much the output score would change if a small perturbation is made to a specific pixel in the input image. By assigning the absolute value of these gradient values to each pixel, a saliency map is obtained that indicates which parts of the input image are most important to the model's decision. Delving deeper into saliency maps and explainability in general, is considered beyond the scope of this thesis.

6. Results

Following the steps described in the previous chapter, we present the following results.

6.1 Model Results

These results refer to the trained CNNs after conducting inference on the 800 images kept aside for test purposes. Table 4 shows the metrics for the models before quantization, while they are still in Floating Point 32-bit format. The training curves for validation accuracy and loss for each base model can be found in the Appendix.

Float32 Model	Accuracy	Precision	Recall	Specificity	F1	MCC
MobileNetv2	91.87%	91.96%	91.87%	98.83%	91.88%	90.72%
GhostNet	84.25%	84.23%	84.25%	97.75%	84.20%	82.01%
GhostNetEdgeTPU	86.00%	86.22%	86.00%	98.83%	91.88%	84.04%
MobileNetEdgeTPUv2	90.75%	90.79%	90.75%	98.67%	90.71%	89.44%
EfficientNetLiteB0	90.37%	90.64%	93.00%	98.82%	90.31%	89.05%
Ensemble Model 1	93.00%	93.02%	93.00%	99.00%	92.99%	92.00%
Ensemble Model 2	91.75%	91.85%	91.75%	98.82%	91.71%	90.59%

Table 4. Model results

After quantizing the models in the integer 8-bit format, we repeat the tests and obtain the following metrics, as shown in Table 5.

Quantized Model	Accuracy	Precision	Recall	Specificity	F1	MCC
MobileNetV2	92.75%	92.77%	92.75%	98.96%	92.73%	91.72%
GhostNet	82.37%	82.66%	82.37%	97.48%	82.35%	79.90%
GhostNetEdgeTPU	85.62%	85.84%	85.62%	97.94%	85.58%	83.61%
MobileNetEdgeTPUv2	90.00%	90.06%	90.00%	98.57%	89.98%	88.58%
EfficientNetLiteB0	89.87%	90.28%	89.87%	98.55%	89.83%	88.50%
Ensemble Model 1	93.12%	93.16%	93.12%	99.01%	93.11%	92.15%
Ensemble Model 2	92.00%	92.07%	92.00%	98.85%	91.98%	90.87%

Table 5. Quantized model results

For the purpose of comparing the models and drawing conclusions, we also present the total memory, as well as the parameter count and the floating-point operations required from each model, as shown in Table 6.

Model	Float Model Size (MB)	Quantized Model Size (MB)	Compiled Model Size (MB)	Parameters	FLOPS
MobileNetV2	21.3	2.5	2.8	2.234.112	612.746.728
GhostNet	36.3	4.2	5.2	3.899.816	266.165.256
GhostNetEdgeTPU	25.0	2.6	3.3	2.429.872	284.026.128
MobileNetEdgeTPUv2	28.4	2.8	3.2	2.528.168	1.031.731.120
EfficientNetLiteB0	30.0	3.7	4.3	3.423.264	781.709.896
Ensemble Model 1	65.0	11.1	12.4	10.221.576	~2.400.000.000
Ensemble Model 2	26.9	5.1	5.6	4.528.586	1.025.303.256

Table 6. Model storage and computational requirements

Last but not least, we evaluate the inference time of each model, as well as the average time required to process one image with the MAPGI framework, on the Coral Edge TPU. The results are shown in Table 7.

Model	EdgeTPU Runtime (ms)
MobileNetV2	14.87
GhostNet	36.73
GhostNetEdgeTPU	20.21
MobileNetEdgeTPUv2	16.63
EfficientNetLiteB0	16.12
Ensemble Model 1	176.39
Ensemble Model 2	22.56
MAPGI Framework	<1

Table 7. Model inference time on the Edge TPU

6.2 Discussion & Conclusions

Before diving into the specific results for each model and performing the comparison between them, we first want to point out the main observation that is shared through all models. Even though the top-1 accuracy of the CNNs tested ranges from 82% up to 93%, all models showcase a top-2 accuracy >=98%. Combining this information with the confusion matrices as shown in Figures 42 and Figure 42, we conclude that the misclassifications between the Esophagitis and Z-line classes are responsible for the significant part of the errors occurred during inference.

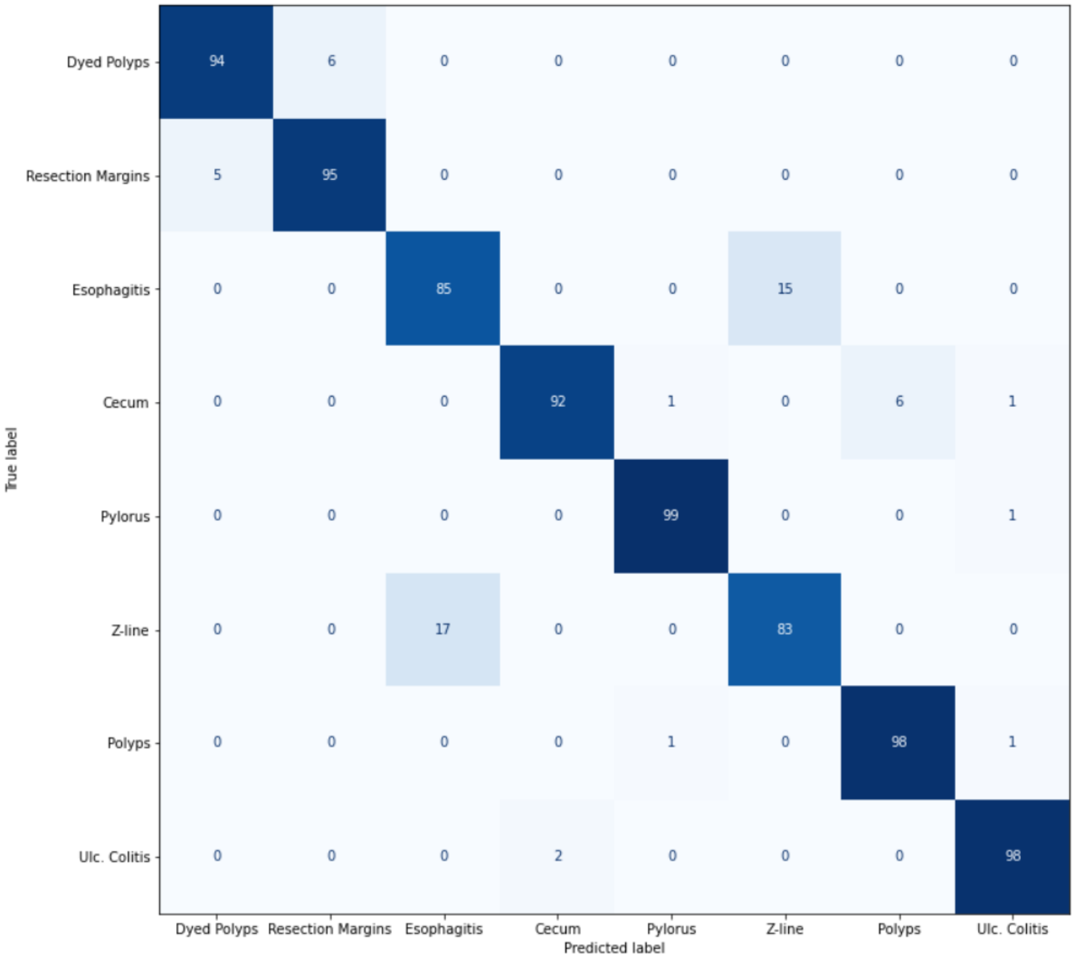
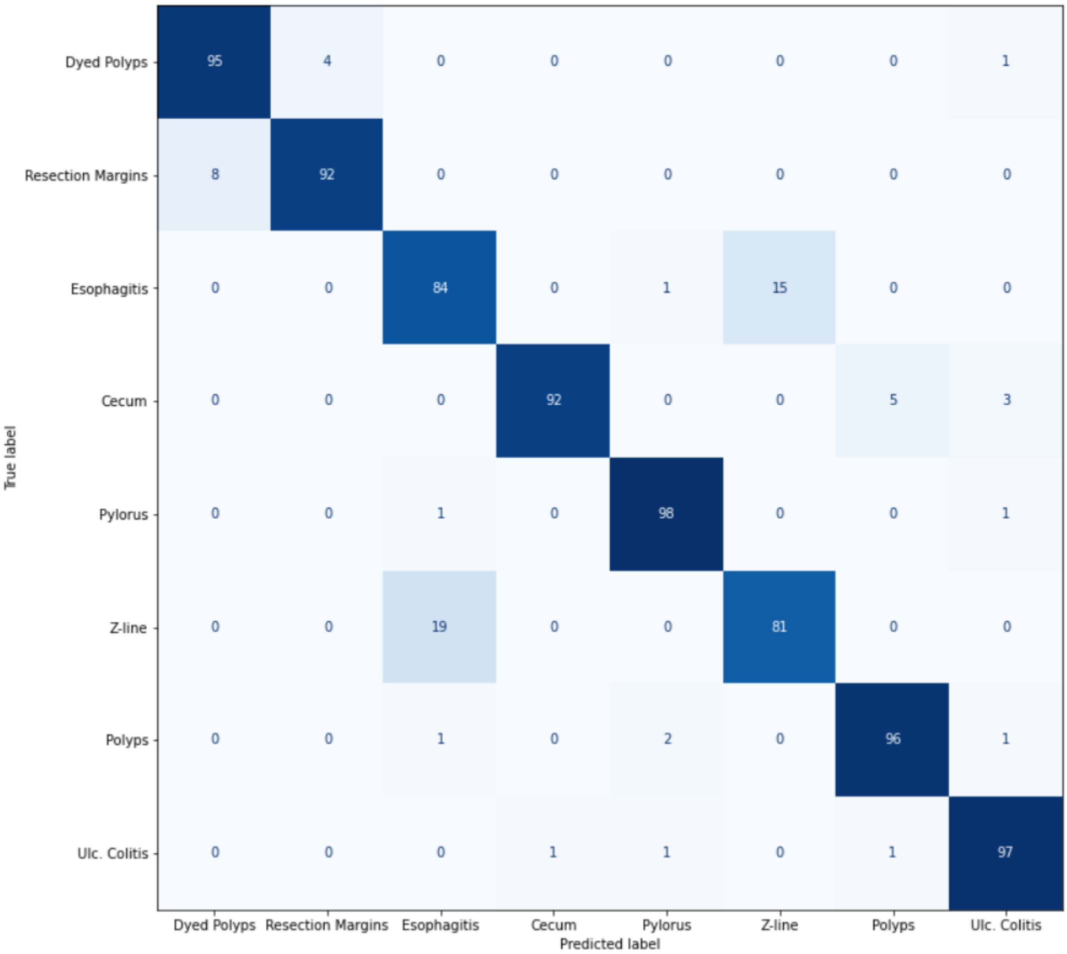


Figure 42 Ensemble Model 2 Confusion Matrix

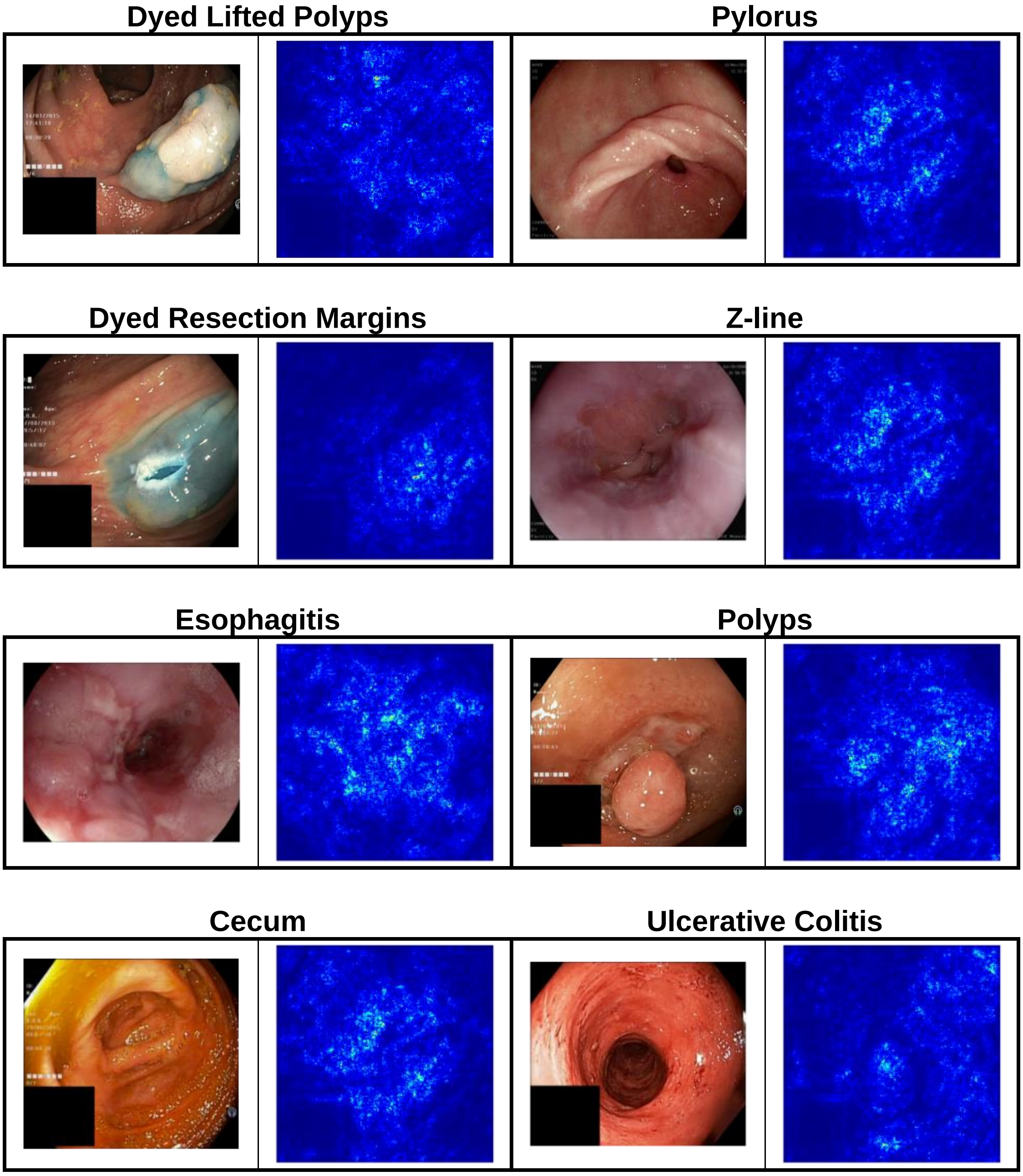


True label \ Predicted label	Dyed Polyps	Resection Margins	Esophagitis	Cecum	Pylorus	Z-line	Polyps	Ulc. Colitis
Dyed Polyps	95	4	0	0	0	0	0	1
Resection Margins	8	92	0	0	0	0	0	0
Esophagitis	0	0	84	0	1	15	0	0
Cecum	0	0	0	92	0	0	5	3
Pylorus	0	0	1	0	98	0	0	1
Z-line	0	0	19	0	0	81	0	0
Polyps	0	0	1	0	2	0	96	1
Ulc. Colitis	0	0	0	1	1	0	1	97

Figure 43 MobileNetV2 Confusion Matrix

This conclusion also highlights the thought process behind the introduction of **Ensemble Model 2**, as an effort to utilize a dedicated network in order to reduce the misclassifications between these aforementioned problematic classes. However, even a MobileNetV2 trained on the 3-class dataset showcased the same behavior and results, leaving us to believe that the learning capacity of these lightweight CNN's cannot further adapt and better learn to differentiate these two classes.

To further understand and visualize the difficulties of Kvasir V2 dataset, we also present the Gradient based Saliency, or Vanilla Saliency, maps produced from MobileNetV2 inference. It is made apparent that model extracts regions of interest different than the human eye. These regions, apart from the center of the pictures where most of the landmarks can be found, also include analysis of the inner walls of the GI tract. A combination of these results in the efficient classification of the test images. Closer depiction also confirms the similarities between the Esophagitis and Z-line gradient heatmaps. It's also made apparent that the CNN struggles to correctly identify regions of interest when it comes to these classes.


Dyed Lifted Polyps

Pylorus

Dyed Resection Margins

Z-line

Esophagitis

Polyps

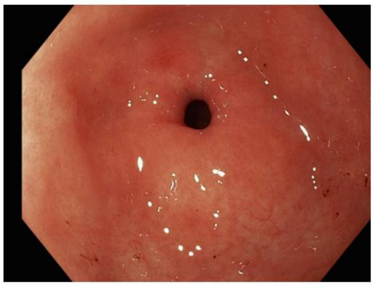
Cecum

Ulcerative Colitis

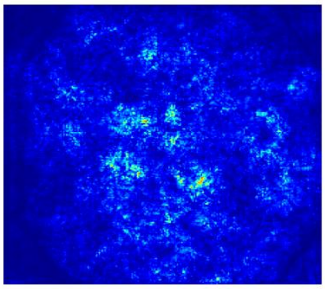
Figure 44. Vanilla Saliency map samples for each class

Most Correctly Classified Class: **Pylorus**

Original Image



Saliency Map



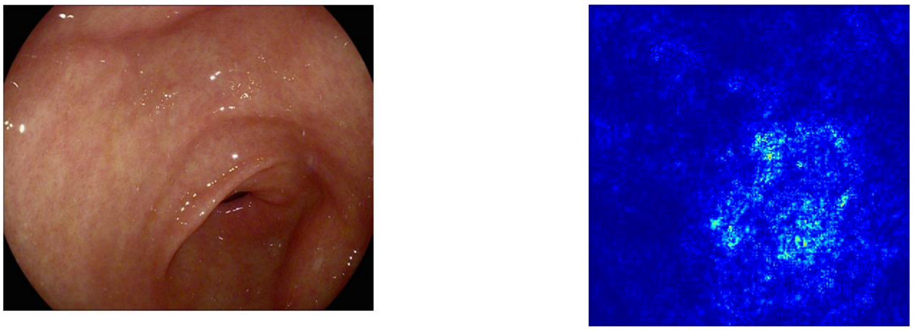
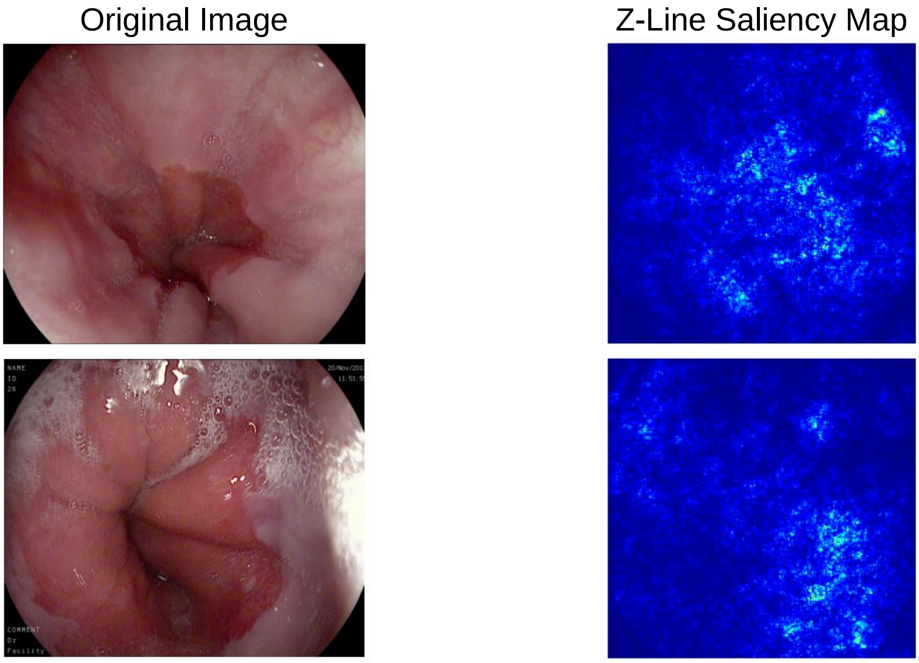


Figure 45. Vanilla Saliency map samples for Pylorus Class

Most Misclassified Classes: **Z-line** – **Esophagitis**

Z-Line

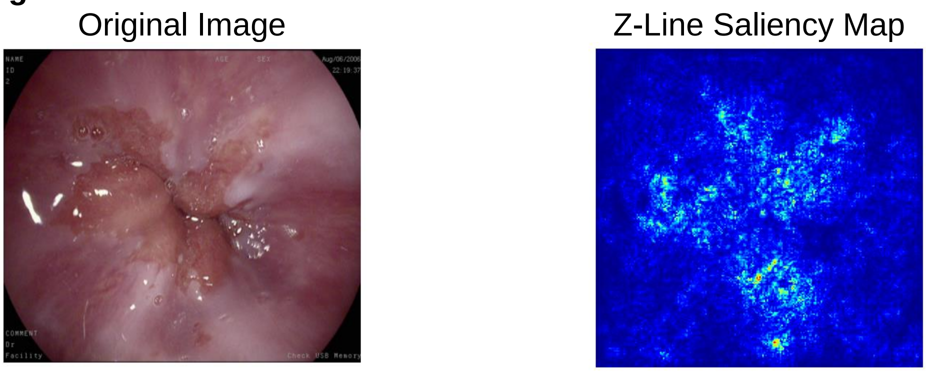


Original Image

Z-Line Saliency Map

Figure 46. Vanilla Saliency map samples for Z-Line Class

Esophagitis



Original Image

Z-Line Saliency Map

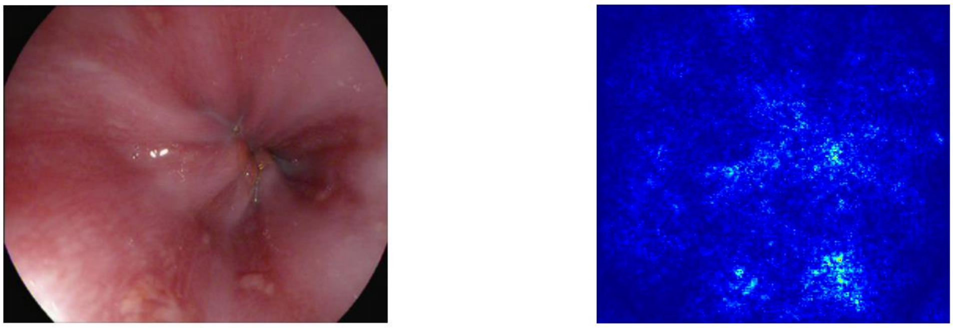


Figure 47. Vanilla Saliency map samples for Esophagitis Class

In terms of specific model results, **Ensemble Model 1** scored the best result with an accuracy of 93.12%. **MobileNetV2** follows closely with a quantized accuracy of 92.75%. It is notable that both Ensemble Model's 1 and MobileNet's accuracy increased after the quantization process, an uncommon occurrence that presents when the rounding of the network's weights after quantization leads to a model that generalizes better on the data. Taking into account the overall model size and inference time, MobileNetV2 is the best performing network. Even though it scored slightly less on most metrics than Ensemble Model 1, it achieved the best inference time at 14.87ms with only 1/5th of the total parameter count, while the ensemble model showcased the worst inference time of 176.39ms. Such difference on runtime occurs due to the fact that the ensemble model's compiled size slightly exceeds the Edge TPU's cache size by 0.4 MB, but comes to show the great effect it has on computation time.

The **MobileNetEdgeTPUv2** and **EfficientNetLiteB0** CNNs showcased similar overall performance, both in accuracy and inference time, but still lacking behind in comparison with MobileNetV2 in terms of parameters-to-runtime ratio. The fact that MobileNetEdgeTPUv2 is slower than the original MobileNetV2 on the Coral Edge TPU, even though it is much faster on a Google Pixel 6 according to its authors, further goes to highlight the importance of matching suitable neural network and hardware architectures.

Ensemble Model 2 performed poorly considering the increase in all aspects, from parameter count to model size and FLOPS, but was an informative tool that provided deep insights into the nature of the dataset.

A case of great interest is that of **GhostNet** and **GhostNetEdgeTPU**. First of all, it is important to mention that these two models are the only original networks that were trained from scratch, as there were no pretrained weights available for the TensorFlow ecosystem, thus falling of behind significantly in accuracy and related metrics. However, these models come with the lowest requirements in FLOPs, requiring less than half of that of MobileNetV2. Unfortunately, this difference does not impact the inference time as expected. The reason behind is that the Ghost Modules utilize depthwise convolutions much more often than the rest of the architectures, and this operation is not as optimized on the edge accelerator than other variants. Finally, the removal of the channel attention mechanisms and the introduction of the ReLU-6 activation function led to significant improvements. The GhostNetEdgeTPU model improved in all metrics in comparison with the original architecture. Most importantly the inference runtime on the accelerator dropped from 36ms down to 20ms, confirming the fact that squeeze-excitation modules are not a good match for this specific hardware architecture.

Last but not least, there is a significant difference between the inference time of our MobileNetV2 and Google Coral's official benchmark. Specifically, the official benchmarks claim a total inference time of 2.6ms, in contrast to our 14.87ms. This is only logical considering the differences in the hardware and the implementation. This gap occurs due to two main factors. First, the Dev Board mini we used utilizes a USB connection between the Edge TPU and the board's CPU, in contrast to the PCI-E connection of the original Board. This adds significant overhead when transferring data from and to the accelerator. The other factor is that the official models are tested using TensorFlow's low level C++ API, whereas we used Python, which lacks in terms of performance.

To summarize some of the conclusions of this thesis:

1. The Kvasir v2 dataset, though small, requires careful preprocessing and handling, in order to minimize the errors presented mainly between the Esophagitis and Z-Line classes.

2. MobileNetV2 was the best model put into test, reaching accuracy levels lower than those presented in literature, but still comparable, and it does so with only a fraction of the parameters and computational effort. With inference time less than 15ms, in addition to the total time required to process one image with the MAPGI framework, which is less than 1ms, we can conservatively expect more than 45 FPS throughput on the Google Coral Dev Board mini.

3. Even though the Google Coral Edge TPU is a powerful edge accelerator that can effectively promote edge applications, like many of its competitors it has not yet reached its full potential in terms of compatibility and optimization regarding the inference of CNNs. Its performance is highly dependent on the correct match between neural network and hardware architecture. While it responds impressively with many convolutional modules, it still lacks proper support for many common operations.

All things consider, we find the results of this thesis encouraging for the promotion of telemedicine, telediagnosis and a variety of real-time computer vision applications on the medical field. The combination of relatively powerful low-cost edge accelerators and the continuous optimization of convolutional neural networks achieved by the technological advancements of the last decade, has reached a level of maturity that will lead to an ever-increasing adoption of computer vision edge applications.

6.3 Future Work

For researchers aiming to improve this work we propose the following:

1. On the context of model accuracy on the Kvasir v2 dataset, one should mainly focus on the two classes responsible for the majority of the misclassifications. As concluded by the results of Ensemble Model 1, the final model metrics are almost equal to the metrics derived by the test classifications performed on the Esophagitis and Z-Line classes. Thus, creating a subset of these two classes, like we did on Ensemble Model 1, and testing the proposed models only on this smaller dataset, can effectively save a lot of training time and produce accurate insights.

2. In terms of model architecture and hardware compatibility, as derived by the comparison of GhostNet and GhostNetEdgeTPU, small changes can have great effect. One "tweak" we have not been able to test is the use of Grouped Convolutions in replacement of the traditional ones, due to the fact that these operations are not yet natively supported by TensorFlowLite. According to Google's official blogpost [28], we have reason to believe that this could lead to significant improvement, and that support for these operations should be implemented in the near future.

Furthermore, for researchers with the goal of extending this work further we present the following ideas:

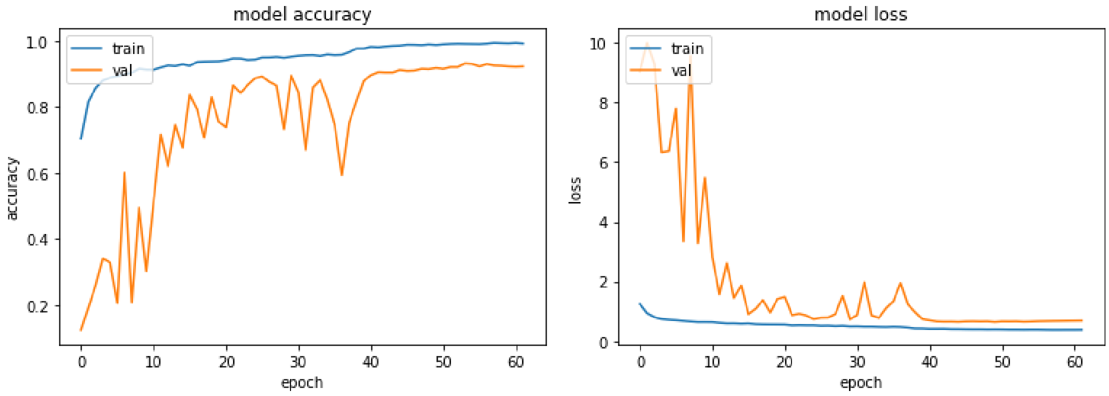
1. Instead of focusing only on models for image classification, this work can be easily extended to include performance benchmarks on image segmentation models. For this purpose, we propose the use of Kvasir SEG, a dataset comprised of 1000 annotated polyp images and their accompanying masks.

2. Last but not least, we highly encourage the performance comparison of the same tasks between different state-of-the-art AI edge accelerators. Given that the development of efficient embedded devices optimized for deep learning inference is still an ever-changing field, this procedure could produce important insights on their differences and lead to generalized conclusions in the way one should approach such tasks.

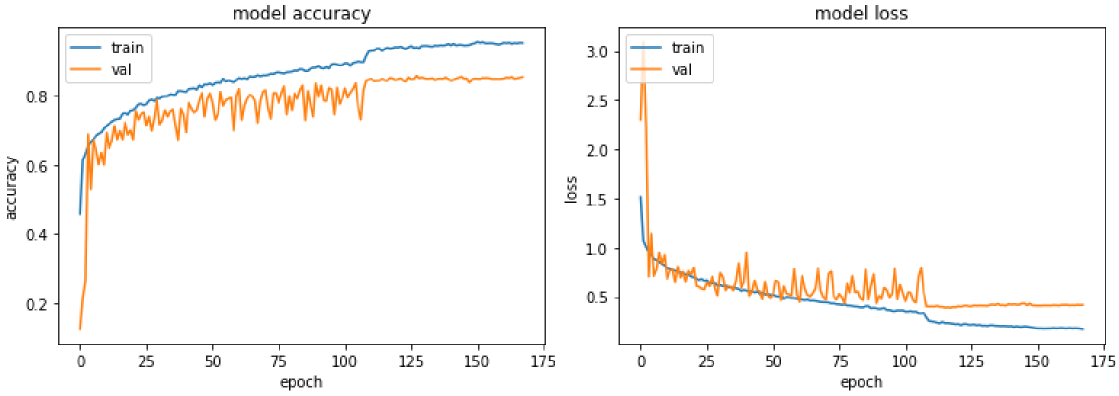
APPENDIX

Training Curves

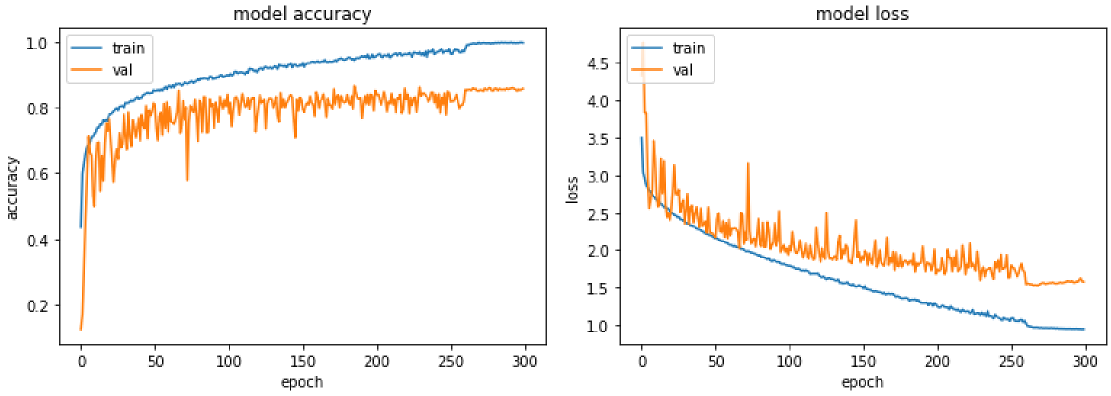
MobileNetV2



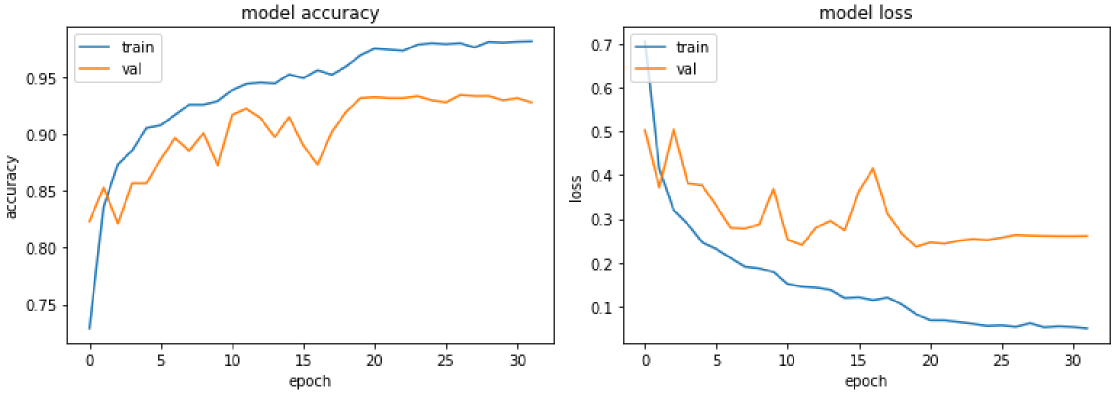
GhostNet



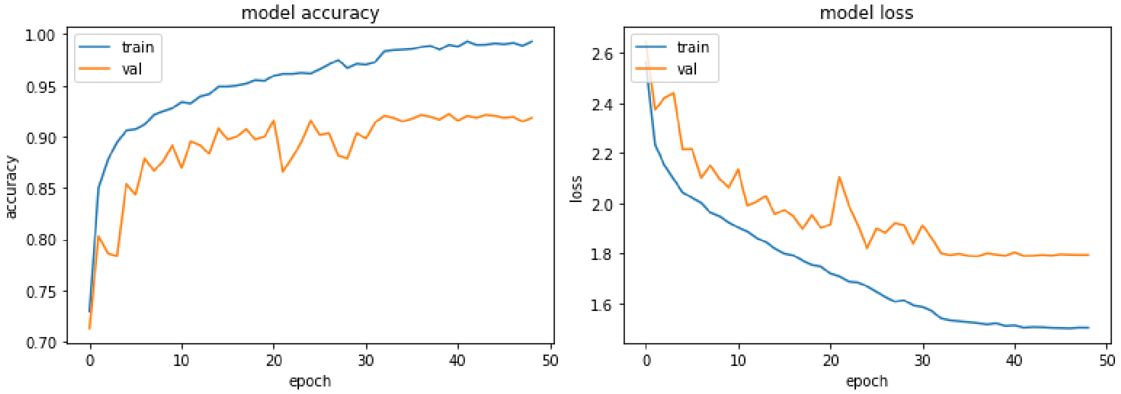
GhostNetEdgeTPU



MobileNetEdgeTPUv2



EfficientNetLiteB0



References

[1] M. Sandler, A. Howard et al, “MobileNetV2: Inverted Residuals and Linear Bottlenecks” IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018

[2] T. Agrawal et al, “SCL-UMD at the Medico Task-MediaEval 2017: Transfer learning-based Classification of Medical Images”, MediaEval’17, 13-15 September 2017

[3] S. Han, S. Mao and W.J.Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding”, ICLR 2016

[4] J. H. Lee et al, “Spotting malignancies from gastric endoscopic images using deep learning”, part of Springer Nature 2019

[5] J. Ribeiro, S. Nobrega and A. Cunha, “Polyps Detection in Colonoscopies”, Procedia Computer Science 196 (2022), p. 477–484

[6] T. Agrawal, R. Gupta and S. Narayanan, “On Evaluating CNN Representations for Low Resource Medical Image Classification”, IEEE ICASSP 2019

[7] C. Gamage et al, “GI-Net: Anomalies Classification in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning”, Moratuwa Engineering Research Conference, 2019

[8] J. Yogapriya et al, “Gastrointestinal Tract Disease Classification from Wireless Endoscopy Images Using Pretrained Deep Learning Model”, Computational and Mathematical Methods in Medicine, v.2021

[9] K. Pogorelov et al, “Kvasir: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection”, MMSys ’17, June 20–23, 2017, Taipei, Taiwan

[10] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning”, arXiv:1603.07285v2

[11] S. Lafraxo and M. El Ansari, “GastroNet: Abnormalities Recognition in Gastrointestinal Tract through Endoscopic Imagery using Deep Learning Techniques”, 2020 8th International Conference on Wireless Networks and Mobile Communications

[12] K. Han, Y. Wang and Q. Tian, “GhostNet: More Features from Cheap Operations”, 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)

[13] T. Cogan, M. Cogan and L. Tamil, “MAPGI: Accurate identification of anatomical landmarks and diseased tissue in gastrointestinal tract using deep learning”, Computers in Biology and Medicine, vol. 111, August 2019

[14] M. Tan and Q.V. Le,” EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”, International Conference on Machine Learning, 2019

[15] S. Voghoei et al, “Deep Learning at the Edge”, 2018 International Conference on Computational Science and Computational Intelligence (CSCI)

[16] Z.M. Lonseko et al, “Gastrointestinal Disease Classification in Endoscopic Images Using Attention-Guided Convolutional Neural Networks”, MDPI, Appl. Sci. 2021

[17] S. Ioffe and C. Szegedy,” Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning, vol. 37, July 2015, pp. 448–456

[18] O.R.A. Almanifi et al, “Automated Gastrointestinal Tract Classification Via Deep Learning and The Ensemble Method”, 21st International Conference on Control, Automation and Systems (ICCAS 2021)

[19] D. Jha, S. Ali et al, “A comprehensive analysis of classification methods in gastrointestinal endoscopy imaging”, Elsevier, Medical Image Analysis, vol.70, 2021

[20] S. A. Magid, F. Petrini and B. Dezfouli, “Image Classification on IoT Edge Devices: Profiling and Modeling”, Cluster Computing, vol 23, issue 2, June 2020, pp. 1025–1043

[21] Andreas M. Kist, “Deep Learning on Edge TPUs”

[22] A. Boroumand et al, “Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks”, 30th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2021

[23] A. Yazdanbakhsh, K. Seshadri, B. Akin et al, “An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks”

[24] S. Poudel, Y.J.Kim, D.M.Vo, “Colorectal Disease Classification Using Efficiently Scaled Dilation in Convolutional Neural Network”, IEEE Acess, May 2020

[25] W.Wang, X.Yang et al, Convolutional-capsule network for gastrointestinal endoscopy image classification, Wiley, December 2021

[26] Öztürk, Ş., Özkaya, U. Gastrointestinal tract classification using improved LSTM based CNN. Multimed Tools Appl 79, 28825–28840 ,2020.

[27] https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html

[28] https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html

[29] G. Menghani, “Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better”, Google Research USA, June 2021

[30] S. Ioffe and C.Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, 2015

[31] Dumoulin, Vincent and Visin, Francesco, “A guide to convolution arithmetic for deep learning”, 2016

[32] Magid, Salma and Petrini, Francesco & Dezfouli, Behnam,” Image classification on IoT edge devices: profiling and modeling”, Cluster Computing. 23. 10.1007/s10586-019-02971-9, 2020

[33] Han, Song & Mao, Huizi & Dally, William. (2016). “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.”

[34] Yazdanbakhsh, Amir & Seshadri, Kiran & Akin, Berkin & Laudon, James & Narayanaswami, Ravi. (2021). “An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks.”

[35] Boroumand, Amirali & Ghose, Saugata & Akin, Berkin & Narayanaswami, Ravi & Oliveira, Geraldo & Ma, Xiaoyu & Shiu, Eric & Mutlu, Onur. (2021). “Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks.” 159-172. 10.1109/PACT52795.2021.00019.

[36] Gholami, Asghar & Kwon, Kiseok & Wu, Bichen & Tai, Zizheng & Yue, Xiangyu & Jin, Peter & Zhao, Sicheng & Keutzer, Kurt. (2018). SqueezeNext: Hardware-Aware Neural Network Design.