Large Filter Low-Level Processing by Edge TPU

Gerald Krell[a] and Thilo Pionteck
Institute for Information and Communication Technology, Otto von Guericke University Magdeburg, Universitätsplatz 2, Magdeburg, Germany

Keywords: Edge TPU, Tensor Flow, Preprocessing, Low-Level Processing, Deep Learning.

Abstract: Edge TPUs offer high processing power at a low cost and with minimal power consumption. They are particularly suitable for demanding tasks such as classification or segmentation using Deep Learning Frameworks, acting as a neural coprocessor in host computers and mobile devices. The question arises as to whether this potential can be utilized beyond the specific domains for which the frameworks are originally designed. One example pertains to addressing various error classes by utilizing a trained deconvolution filter with a large filter size, requiring computation power that can be efficiently accelerated by the powerful matrix multiplication unit of the TPU. However, the application of the TPU is restricted due to the fact that Edge TPU software is not fully open source. This limits to integration with existing Deep Learning frameworks and the Edge TPU compiler. Nonetheless, we demonstrate a method of estimating and utilizing a convolutional filter of large size on the TPU for this purpose. The deconvolution process is accomplished by utilizing pre-estimated convolutional filters offline to perform low-level preprocessing for various error classes, such as denoising, deblurring, and distortion removal.

1 INTRODUCTION

In this paper, we introduce the utilization of Edge Tensor Processing Units (Edge TPUs) for the low-level processing of video streams. This task typically demands significant computational resources and is often performed in close proximity to the sensor within the processing pipeline, before any subsequent high-level image processing can occur.

Edge computing aims to process and store data in close proximity to its sources (Merenda et al., 2020; **?**). According to (Yazdanbakhsh et al., 2021), Tensor Processing Units (TPUs) were initially developed to accelerate machine learning inference within data centers and later extended to support machine learning training. Edge TPUs have been introduced in a limited number of products as inference accelerators at the edge, positioned near sensors or databases.

Typical applications of Edge TPUs, as highlighted by (Coral, 2021a), include image classification, object detection, semantic segmentation, pose estimation, and speech recognition.

One significant challenge when applying the Edge TPU to non-standard applications is the limitation of its function set. This limitation is illustrated in Figure 1. While TensorFlow encompasses a wide array of features for deep learning training and inference, TensorFlow Lite (Google, 2021) is designed to optimize models for less resource-intensive implementations, such as mobile applications or embedded devices like the Edge TPU.

TensorFlow Lite offers a subset of the TensorFlow API specifically tailored for machine learning on mobile and edge devices. It includes speed-optimized functions for hardware implementation and quantization techniques to represent data and model parameters as integers.

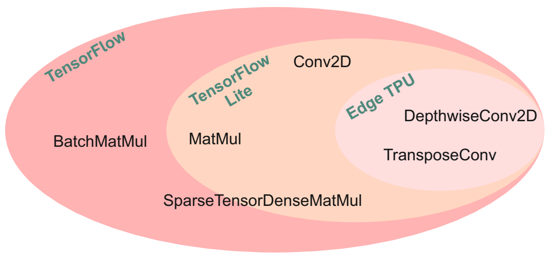


Figure 1: TensorFlow API function set and subsets with function candidates for Large Filter low-level processing

The function set of the Edge TPU is even smaller which means that non-supported functions are outsourced and executed on the hosting CPU. An even smaller set is supported by the EdgeCompiler (Coral, 2021b)

[a] https://orcid.org/0000-0002-5526-0446

Krell, G. and Pionteck, T.
Large Filter Low-Level Processing by Edge TPU.
DOI: 10.5220/0012370500003660
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2024) - Volume 3: VISAPP*, pages 464-473
ISBN: 978-989-758-679-8; ISSN: 2184-4321
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

Hence, we illustrate which functions can be employed for low-level processing using the Edge TPU. Figure 1 offers a comparison between the extensive function set of TensorFlow and TensorFlow Lite and the highly limited set supported by the Edge Compiler. We identify potential functions suitable for our targeted image low-level processing.

Regrettably, some promising functions, such as MatMul, BatchMatMul, or SparseTensorDenseMatMul for direct matrix operations, are not available for the Edge TPU. TensorFlow provides the Conv2d operation, commonly used in Convolutional Neural Networks (CNNs). However, the Edge TPU only supports the separable version, DepthwiseConv2D, which should suffice for most filtering tasks.

In the realm of image low-level processing, convolution serves as the mathematical foundation for linear filtering. This entails convolving a one or more-dimensional image with a corresponding one or more-dimensional filter kernel. When working with separable filter kernels, separable filtering is performed in each dimension. This approach helps conserve mathematical operations and, as a result, accelerates processing speed.

To achieve this, Deep Learning (DL) models are developed to implement deconvolution filters. These models are estimated through a training process that utilizes original and degraded images, taking into account even unknown properties of the image acquisition system.

Our research demonstrates that these deconvolution models can be effectively implemented as finite impulse response (FIR) filters on Edge TPU hardware, as provided by the Coral USB accelerator (Coral, 2022).

The paper is structured as follows:

First, it begins by presenting the tools employed to apply the Edge TPU for preprocessing video streams. This is followed by an exploration of the processing power available in the matrix multiplication unit of the Edge TPU, in comparison to the requirements of the targeted preprocessing model.

Next, the paper delves into the mathematical formulation of convolution to model the image degradation process. It introduces the deconvolution model, which is utilized to estimate the restoration parameters necessary for low-level processing. The paper also elucidates how model parameters for restoration can be derived from undistorted and distorted example images.

The final section of the paper showcases experiments related to model estimation and its execution on the Edge TPU using sample data. Additionally, it evaluates the results of the restoration process in relation to model parameters. The paper also demonstrates the advantages of using large filter sizes in different aspects.

2 RELATED WORK

The stages of low-level processing, such as filtering and noise suppression, although not explicitly mentioned, are of paramount significance in the context of most video stream analysis applications. To address this, (Basler, 2021a) has incorporated an image signal processor into the digital camera video pipeline for low-level processing. This processor can be implemented either as a customized chip directly behind the camera sensor or as an additional component within a processor (Basler, 2021b), specially designed for machine learning and computer vision tasks.

In their work, (Sun and Kist, 2022) provides a comprehensive overview of the properties and application areas of Edge TPUs while also discussing their general limitations.

Additionally, (Abeykoon et al., 2019) successfully ported networks for image restoration to the Edge TPU, showcasing its versatility.

Moreover, (Zeiler et al., 2010) introduces Deconvolutional Networks, a valuable approach for unsupervised construction of hierarchical image representations, which find applications in low-level tasks like denoising and feature extraction for object recognition.

Furthermore, research, such as that presented in (Lecun et al., 1998), demonstrates that hand-crafted feature extraction can be advantageously replaced by carefully designed learning machines that can operate directly on pixel images, as exemplified in character recognition within the scope of (Lecun et al., 1998).

In their study, (Markovtsev, 2019) delve into the interaction between TensorFlow and the Edge TPU. They discuss the availability of convolutional operations and fully connected neural inference operations on the Edge TPU, taking advantage of the robust arithmetic hardware of the TPU. Furthermore, they demonstrate how motion blur can be simulated using the Edge TPU with DepthwiseConv2d.

In a different research effort, (Yazdanbakhsh et al., 2021) reported on the inference of 234K distinct convolutional neural networks applied to three different categories of Edge TPU accelerators. The research involved measuring latency under varying calculation graph depth and width, and it revealed high inference accuracy.

Additionally, (Civit-Masot et al., 2021) compares the application of Edge TPUs to eye fundus image segmentation with a Single Board Computer lacking deep learning acceleration. The study demonstrates that machine learning-accelerated segmentation can achieve processing times below 25 ms per image, underscoring the efficiency of Edge TPU-based solutions.

3 TOOLS

3.1 Workflow

While the TPU's CISC instruction set consists of only a limited number of instructions (Harald Bögeholz, 2017), direct programming of the Edge TPU (ETPU) is not feasible due to the fact that the available ETPU compiler and its controlling shared library are proprietary (Markovtsev, 2019). Consequently, ETPU programming typically relies on utilizing existing TensorFlow Lite models in conjunction with the ETPU compiler.

Instead of directly programming the ETPU, the standard practice involves compiling established TensorFlow Lite models using the Edge TPU compiler to enable their execution on the device. Notably, existing TensorFlow models need to be converted into the TensorFlow Lite format, which is constrained by quantization to unsigned byte integers. It's worth noting that direct matrix multiplication is not supported by the Edge TPU, despite the presence of a matrix multiplication unit in the Edge TPU hardware, which might suggest suitability for matrix operations.

The workflow for converting a TensorFlow model for the Edge TPU is the following:

1. Model generation by TensorFlow (computation graph)
2. Convertion to TF Lite format (flatbuffers, quantization uint8)
3. Invokation of edgetpu_compiler
4. Edge TPU ops delegate, invoking the new model via the interpreter

In our experiments, we employed Matlab on a PC to estimate the finite impulse response (FIR) filter coefficients for deconvolution using sample images, as described in the experiments section. Subsequently, these restoration coefficients were used in the Python framework on a Linux PC with the Edge TPU as an accelerator to create and compile corresponding TensorFlow Lite (TF Lite) models for execution on the Edge TPU.

3.2 Hardware

Both the Coral DevBoard and the USB accelerator incorporate the same Edge TPU and have been compared, as shown in Figure 2. The former is capable of direct integration with a camera sensor and operates independently with its own Linux module.

The latter, on the other hand, can function as a coprocessor for a Linux PC or a Raspberry PI. Video streams can be transmitted via a network connection. When it comes to developing models, working with the USB accelerator is somewhat more convenient. This is because models can be compiled directly by the host and then transferred to the coprocessor.

In contrast, the DevBoard's operating system is quite limited and not ideal for model estimation or compilation for the Edge TPU. These tasks must be performed on a separate host computer, and the computed models must be subsequently transferred to the DevBoard in an additional step.

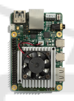
	DevBoard	USB accelerator
		
advantages	integrated peripherals, direct sensor connection (camera on board), no load on host	direct connection to host (e.g. PC or Raspberry PI)
disadvantages	no compilation of models, limited functions of mendel linux	no integrated peripherals

Figure 2: Comparison between USB accelerator and DevBoard.

4 PERFORMANCE CONSIDERATIONS

In (Harald Bögeholz, 2017), it is noted that Tensor Processing Units (TPUs) have a theoretical upper limit of 92 T (tera) operations per second. This high processing capacity opens the door to intriguing applications that require real-time processing of large data streams.

On the other hand, the Edge TPU is designed for mobile devices to run TensorFlow Lite models and provides a maximum processing power of 4 T operations per second (Wikipedia contributors, 2021). While this is significantly less powerful compared to TPUs, it is still a substantial amount of processing power, primarily dedicated to model inference tasks.

The proposed low-level processing, based on deconvolution, aims to be implemented through deep learning (DL) models that realize finite impulse response (FIR) filters on the Edge TPU. Given the Edge TPU's robust arithmetic hardware, a critical question arises regarding the size of FIR filters that can be effectively implemented.

Let M_{ETPU} denote the processing power of one Edge TPU clock cycle, representing the number of multiplications-accumulations (MACs) that the Edge TPU can compute per clock cycle. Assuming a relationship R between the Edge TPU clock and the pixel clock, the maximum available processing power of the Edge TPU for real-time processing of one pixel is defined as:

$$M_P^{max} = M_{\text{ETPU}} \cdot R \quad . \tag{1}$$

Depending on the specific Edge TPU model and the resolution of the video stream, the value of R may typically fall within the range of 10. We can now compare this *provided* processing power M_P^{max} for one pixel with the *required* processing power M_P^{req} for filtering one pixel, both in the case of non-separable and separable filtering.

4.1 Non-Separable Filter

Given a filter size of F_s in two dimensions for a non-separable filter, and considering C color channels, the required computational costs for processing one pixel are defined as:

$$M_P^{req} = F_s^2 \cdot C^2 \ . \tag{2}$$

The assumption of two dimensions is motivated by the Edge TPU's primary focus on processing 2-D image streams. It is also assumed that the horizontal and vertical filter sizes are equal, which is often the case for many tasks and is sufficient for a rough estimation. The square of channel numbers accounts for cross-couplings between the channels, which consider mutual influences between the channels. Such filters can handle color errors at edges and even facilitate color space rotation when needed.

We find the maximum filtersize F_s^{max} by rearranging equation 2 according to F_s and substituting M_P^{req} by M_P^{max} and get

$$F_s^{max} = \left\lfloor \sqrt{\frac{M_P^{max}}{C^2}} \right\rfloor = \left\lfloor \sqrt{\frac{M_{\text{ETPU}} \cdot R}{C^2}} \right\rfloor \tag{3}$$

Given that $M_{\text{ETPU}} = 4\text{Ki}$ for an Edge TPU matrix multiplication unit with a size of 4096, and assuming that the Edge TPU clock is 10 times faster than the pixel clock ($R = 10$), we can calculate the maximum filter size F_s^{max} as follows: $F_s^{max} = \sqrt{\frac{4\text{Ki}\cdot 10}{3^2}} = 67$.

If we don't need to consider cross-coupling between color channels in the filtering process and can treat the color channels independently, the processing costs specified in (2) can be reduced to:

$$M_P^{req-wo} = F_s^2 \cdot C \tag{4}$$

This represents the required MACs per pixel without cross-coupling, providing the processing power needed when there's no coupling between color channels.

For a filter without cross-coupling, the maximum filter size approximately doubles:

$$F_s^{max-wo} = \left\lfloor \sqrt{\frac{M_{\text{ETPU}} \cdot R}{C}} \right\rfloor = \left\lfloor \sqrt{\frac{4\text{Ki} \cdot 10}{3}} \right\rfloor = 116 \ . \tag{5}$$

Indeed, filter sizes of this magnitude are quite comfortable and versatile. They are not only suitable for tasks like noise cancellation and deblurring but can also handle geometric transformations to compensate for lens distortions or perform other coordinate manipulations effectively. This level of processing power opens up a wide range of applications in image enhancement and correction, making it a valuable capability for the Edge TPU.

4.2 Separable Realization

For separable filters, the situation further improves as the calculation costs reduce proportionally from F_s^2 to $2F_s$. This results in the maximum possible separable filter size given by:

$$F_s^{max-sep} = \frac{M_{\text{ETPU}} \cdot R}{2C^2} = \frac{4\text{Ki} \cdot 10}{2 \cdot 3^2} = 2275 \tag{6}$$

This allows for significantly larger separable filter sizes when considering cross-couplings, and even about three times larger filter sizes for filters without cross-coupling.

With the substantial filter sizes achievable, enabling full coupling over the entire image size by the filter becomes realistic for typical image resolutions. If these filters are available as space-variant filters, they could be effectively employed for various error correction tasks, including compensation of distortions that are addressed through warping. This level of processing capacity would offer sufficient resources for what can be considered as "convenient warping" in combination with filtering, providing a versatile and powerful solution for a range of image processing and correction needs.

On the other hand, it's important to note that separable filtering is not well-suited for certain tasks, such as the removal of rotational blur or certain types of distortions. Separable filters are most effective for filters with simple spatial dependencies in the horizontal and vertical directions. For filters with complex, non-axial dependencies between pixels, separable filters may not be as suitable, and other methods or more complex filter designs may be required to address these tasks effectively.

5 CONVOLUTION MODEL

The continuous formulation of convolution is represented by the equation:

$$g(x) = \int_{-\infty}^{\infty} h(\xi) f(x-\xi) d\xi = h * f\ . \tag{7}$$

Here, the convolution result at a given position, denoted as $g(x)$, is obtained through the infinite integration of the product of functions h and the shifted function f over the variable ξ.

In digital signal processing, we work with a discrete and limited form of convolution. We represent the signal f as a vector $\vec{f}$ containing discrete samples, and the filter as a vector $\vec{h}$, resulting in the convolution operation denoted as $*$. This discrete convolution yields the result $\vec{g}$ in vector representation:

$$\vec{g} = \vec{h} * \vec{f} \tag{8}$$

Here, the resulting convolution samples in the vector $\vec{g}$ are obtained through the convolution operation $*$ between the vectors $\vec{h}$ and $\vec{f}$, where $\vec{h}$ contains the coefficients of the filter h and $\vec{f}$ contains the samples of the signal f.

When performing convolution over a filter with M sample points, where the index is denoted as j, the discrete convolution result is calculated using the following sum for each sample point with index i:

$$g_i = \sum_{j=0}^{M-1} h_j f_{i-j}\ . \tag{9}$$

This calculation is carried out for N sampling points, where i varies within the range $[0, N-1]$. However, when choosing the range $i = 0, 1, 2, \ldots, N-1$, assumptions must be made for negative indexes of f. One common approach is to use zero padding, which means that any negative indices of f are assumed to be zero.

We can represent the convolution as the multiplication of a matrix by a vector, as defined in (10). This can be expressed as:

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_i \\ \vdots \\ g_{N-1} \end{bmatrix} = \begin{bmatrix} h_{M-1} & \cdots & h_1 & h_0 & 0 & & \cdots & 0 \\ 0 & h_{M-1} & \cdots & h_1 & h_0 & 0 & \cdots & 0 \\ & & & & \vdots & & & \\ 0 & \cdots & 0 & h_{M-1} & \cdots & h_1 & h_0 & 0 \\ 0 & & \cdots & 0 & h_{M-1} & \cdots & h_1 & h_0 \end{bmatrix} \begin{bmatrix} f_{-(M-1)} \\ \vdots \\ f_{-1} \\ f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix} \tag{10}$$

.

or shorter

$$\vec{g} = \mathbf{H}_{ex} \vec{f}_{ex} \tag{11}$$

Here, $\vec{g}$ and $\vec{f}_{ex}$ are vectors that contain discrete values and can be used to represent result and input images, respectively, in the context of image processing.

To obtain a result vector with N elements, the input vector f must be extended by $M-1$ elements above to form $\vec{f}ex$ since, in the convolution equation (9), otherwise, the index range of f would fall below the range of i. Similarly, the matrix $\mathbf{H}$ is extended on the left by $M-1$ columns to become $\mathbf{H}_{ex}$.

In the field of signal processing, various strategies are discussed for implicit extension at the borders of limited discrete signals, such as periodic or mirrored extension. A practical assumption, also applicable for hardware implementations as a data stream, is to set these margin values to zero, a technique known as zero padding. In this case, $\vec{f}$ is extended with zeros for negative indices.

The convolution matrix $\mathbf{H}$ is a sparse and circulant Toeplitz matrix with a size of N by $N+M-1$, where each row is a right circular shift of the row above it, and the elements in the main and side diagonals are equal. However, this characteristic does not hold for space-variant systems in which the filter h is dependent on the location x.

When setting the border elements to zero, the matrix equation (10) can be rewritten as follows:

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_i \\ \vdots \\ g_{N-1} \end{bmatrix} = \begin{bmatrix} h_0 & & & & \\ h_1 & h_0 & & 0 & \\ & & \vdots & & \\ h_{M-1} & \cdots & h_1 & h_0 & 0 \\ & & \vdots & & \\ 0 & h_{M-1} & \cdots & h_1 & h_0 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix} \tag{12}$$

or

$$\vec{g} = \mathbf{H}\vec{f}\ . \tag{13}$$

The matrix $\mathbf{H}$, when border elements are set to zero, forms a non-circulant square Toeplitz matrix. When dealing with two or more dimensional signals, as is common in image processing, the data needs to be vectorized (Andrews and Hunt, 1977). Although a general matrix multiplication is not available in the supported Edge TPU functions, the specific case of multiplying a Toeplitz matrix (filter) by a vector (for 1D or 2D images) is provided by convolution functions. This enables efficient processing of multidimensional signals and images using the Edge TPU's capabilities.

$$\begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_i \\ \vdots \\ g_{N-1} \end{bmatrix} = \begin{bmatrix} f_0 & & 0 \\ f_1 & f_0 & 0 \\ & & \\ & \vdots & \\ & & \\ f_{N-1} & \cdots & f_{N-M} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{M-1} \end{bmatrix} \tag{14}$$

or

$$\vec{g} = \vec{f} * \vec{h} = \mathbf{F}\vec{h} \tag{15}$$

Utilizing convolution to model the process of image degradation due to imperfections in the optical system of image acquisition is a common approach in image processing. Typically, additional noise needs to be considered, resulting in the convolution equation:

$$\vec{\tilde{g}} = \vec{g} + \vec{n} \tag{16}$$

Deconvolution aims to find a compensation for convolution using a function represented by a discrete vector $\vec{\tilde{h}}$ in such a way that it minimizes the difference between the convolved result $\vec{\tilde{g}} * \vec{\tilde{h}}$ and the original image $\vec{f}$. Mathematically, deconvolution seeks to minimize the following objective:

$$\left\| \vec{\tilde{g}} * \vec{\tilde{h}} - \vec{f} \right\| \to \min . \tag{17}$$

Deconvolution, even though it is a linear system, is often an ill-conditioned problem, which implies that it may lack a unique or stable solution. To address this challenge, we investigate approaches to finding well-posed solutions for estimating model parameters in a broader context. These techniques are applied to achieve reliable and stable deconvolution results, mitigating the ill-conditioned nature of the problem.

6 EXPERIMENTS

6.1 Training Data Generation

As an example, the motion blur filter of (Markovtsev, 2019) was used to generate a degraded version of a test image (see Figure 3)

For 2-dimensional image filtering, both the image and the filter need to be vectorized to be used in the matrix equation. If we consider $\vec{f^o}$ as the vectorized original image and $\vec{h^b}$ as the blurring vector obtained by vectorizing the filter kernel, we can represent the motion-blurred image in vectorized form as:

$$\vec{g^b} = \vec{f^o} * \vec{h^b} \tag{18}$$

This notation is similar to (13), which also deals with vectorized image and filter data, assuming that the boundary elements are set to zero. In this specific example, the filter kernel, as shown in Figure 3a, defines $\vec{h^b}$. The kernel size is 21 by 21, resulting in $N = 21 \cdot 21 = 441$ MAC operations per pixel. To simulate a motion blur of approximately 10 pixels in the horizontal direction, most elements of $\vec{h}$ are set to zero, except for the values located on a horizontal line in the vertical middle of the right half. Convolution of the input $\vec{g}$, provided by the image from Figure 3b, yields a simulated motion-blur result $\vec{g}$ as shown in Figure 3c.

6.2 Estimation of Model Parameters

For deconvolution, the goal is to find a deconvolution filter vector $\vec{\tilde{h}}^r$ of length F_s to reconstruct a restored vector, denoted as

$$\vec{\tilde{f}} = \vec{g^b} * \vec{\tilde{h}}^r \tag{19}$$

using the motion-blurred image vector $\vec{g^b}$. The objective is to make $\vec{\tilde{f}}$ deviate as little as possible from the original image $\vec{f}$, and the error signal is represented by the error vector:

$$\vec{e} = \vec{f^o} - \vec{\tilde{f}} \tag{20}$$

The objective is to minimize the power of this error signal, making the error as small as possible. This process aims to reverse the blurring, noise, and other degradations to restore the image as faithfully as possible. The goal is to achieve a high-quality restoration of the original image.

In practice, we aim to minimize the mean squared deviation between the original and the restoration, where || indicates the norm:

$$\|\vec{e}\|^2 = \left\| \vec{f^o} - \vec{g^b} * \vec{\tilde{h}}^r \right\|^2 \to \min . \tag{21}$$

Expressed as a matrix equation, we have:

$$\left\| \vec{f^o} - \hat{\mathbf{G}}^b \vec{\tilde{h}}^r \right\| \to \min . \tag{22}$$

The restoration filter vector, $\vec{\tilde{h}}^r$, can be estimated by solving a linear equation system with F_s unknowns in the least mean square sense. This is achieved by converting equation (19) into a matrix form:

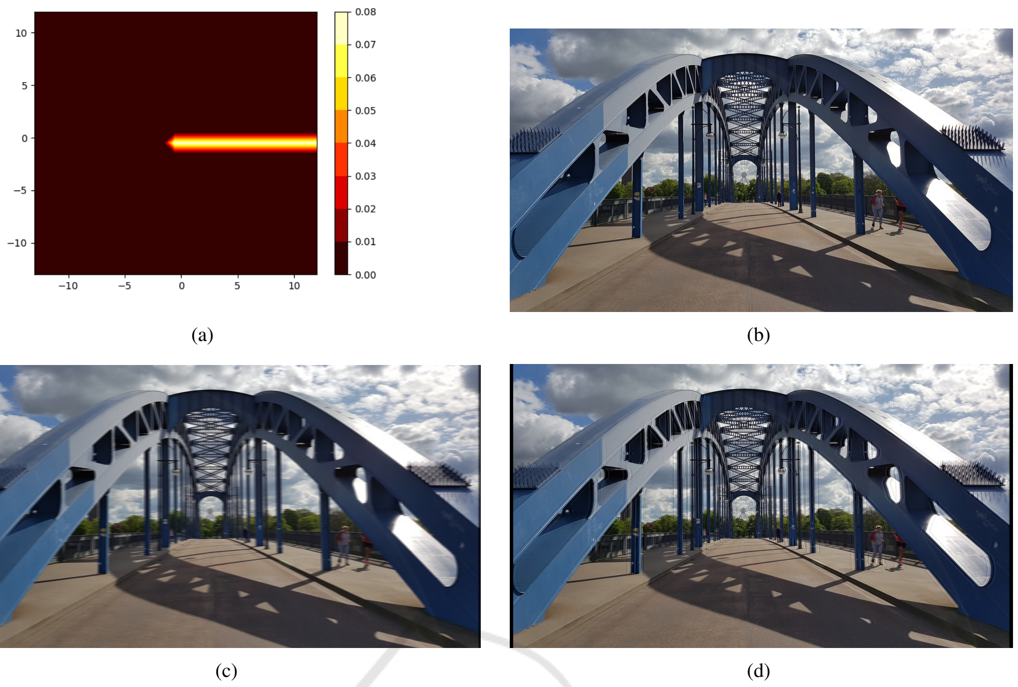


Figure 3: Two-dimensional image data for motion blur generation: 3a Kernel: squared filter kernel for motion blur simulation producing blurring filter vector $\vec{h}^b$; 3b Input of convolution: original image producing $\vec{f}^o$; 3c Output of convolution: motion blur simultation producing $\vec{g}^b$; 3d Deconvolution Result $\hat{O}$ for $F_s = 27$;

$$\begin{bmatrix} f_{F_s-1} \\ f_{F_s} \\ \vdots \\ f_{N-1} \end{bmatrix} = \begin{bmatrix} g^b_{F_s-1} & \cdots & g^b_1 & g^b_0 \\ g^b_{F_s} & \cdots & g^b_2 & g^b_1 \\ & & \vdots & \\ g^b_{N-1} & & \cdots & g^b_{N-F_s} \end{bmatrix} \begin{bmatrix} \hat{h}^r_0 \\ \hat{h}^r_1 \\ \vdots \\ \hat{h}^r_{F_s-1} \end{bmatrix} \tag{23}$$

or

$$\vec{f}^o = \vec{g}^b * \vec{\hat{h}}^r = \hat{\mathbf{G}}^b \vec{\hat{h}}^r \, , \tag{24}$$

where $\hat{\mathbf{G}}^b$ is the equalization matrix of size $n_r = N - F_s + 1$ rows by F_s columns assuming that we generate sample data from all all the elements of $\vec{g}^b$. The rows of this matrix consist of pixel-wise shifted sections of $\vec{g}^b$. In this case, there can only be a solution for $\vec{\hat{h}}^r$ in the least mean square sense because we assume $n_r \gg F_s$, meaning that we have many more equations than unknowns. To solve equation (21), we first multiply equation (24) by the transpose (T) of $\hat{\mathbf{G}}^b$ from the left:

$$\left(\hat{\mathbf{G}}^b\right)^T \vec{f}^o = \left(\hat{\mathbf{G}}^b\right)^T \hat{\mathbf{G}}^b \vec{\hat{h}}^r \tag{25}$$

$$\tag{26}$$

We can solve for the restoring filter as follows:

$$\vec{\hat{h}}^r = \hat{\mathbf{G}}^{b+} \vec{f}^o \, . \tag{27}$$

Here, $\hat{\mathbf{G}}^{b+}$ represents the pseudo-inverse of $\hat{\mathbf{G}}^b$. Assuming that $\hat{\mathbf{G}}^{b+}$ has at least F_s linearly independent rows, we can estimate the restoration using equation (19), resulting in a restored image

$$\vec{\hat{f}} = \vec{f}^o - \vec{e} \, . \tag{28}$$

that deviates from the original by the error vector $\vec{e}$.

6.3 Running the Inference on the Edge TPU

In the context of Deep Learning, $\vec{\hat{f}}$ can be viewed as the result of inferring the model $\vec{\hat{h}}^r$ using the input vector $\vec{g}^b$, as illustrated in Figure 4. Model estimation is carried out on the host computer using samples from both the ideal and restored image streams. Inference is then performed on the Edge TPU to compensate for the errors resulting from image degradation.

This approach estimates the restoring filter in a manner similar to an inverse filter. However, because equation (22) usually has many more rows than unknowns, a solution in the least mean square sense can be found. This approach takes typical image noise into consideration, resulting in a more stable and robust result compared to directly inverting a degrading filter matrix $\mathbf{H}$.

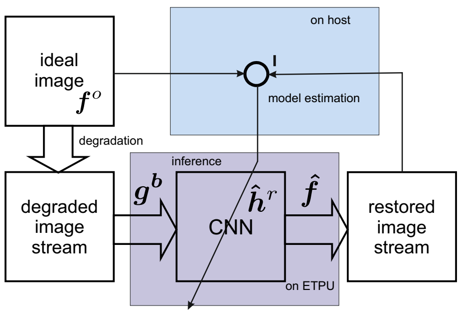


Figure 4: Dimensioning the CNN model for removal of image errors

Based on the training data (see section 6.1) models of different filter sizes F_s have been estimated by (27) using Matlab for compensating the motion blur. Figure 3d show one of the results for $F_s = 27$. In the practical implementation of the model estimation, not the complete image is maybe necessary in (27) and subimages of the whole image can be used. This can also reflect specially varying behaviour.

The estimated Filter models first converted into a quantized TFLite representation and then compiled for the Edge TPU according to (Markovtsev, 2019). The inference has then be done on the USB accelerator with the training input image Figure 3c. The result corresponds to the expected inference result of Figure 3d with little deviations caused by quantization of the model parameters, but below a visual threshold.

6.4 Influence of Filter Size

In order to investigate the impact of filter size onto the the obtained enhancement we investigated the gain of PSNR (Peak Signal to Noise Ratio) by applying the deconvolution filter model for restoration:

$$G = \text{PSNR}_w - \text{PSNR}_{wo} \tag{29}$$

where $\text{PSNR}_{wo} = 20\frac{I_{\max}}{||\vec{f}^o - \vec{g}^b||}\text{dB}$ is the PSNR of the input of the deconvolution and $\text{PSNR}_w = 20\frac{I_{\max}}{||\vec{e}||}\text{dB}$ the PSNR of the restoration with $I_{\max} = 255$ the maximum pixel value.

First we applied deconvolution to the motion blurred image $\vec{g}^b$ and changed the horizontal F_s in a range $[1, 15]$.

Figure 5 shows the dependency of G on the filter size F_s^h. It is obvious that a minimum filter size of about $F_s^h = 10$ is required in order to obtain a meaningful improvement. Above filter size 30 the growth of the gain is decreasing rapidly. This behavior can be explained by the original motion blur of about 7 pixels which should be in the same range as F_s.

Secondly we additionally superimposed normal distributed noise of 40 dB. We kept the horizontal filtersize at 27 which could be considered as an optimal filter size for motion blur compensation in horizontal direction and increased the vertical filter size steps for $F_s^v = 1, 3, \cdots, 15$. It is visible that in the current situation of noise and blur a filter size of about 7 is sufficient in order to compensate additionally for the noise degradation. Together with horizontal filter size we come to a total 2 dimensional filter size of $F_s = F_s^v F_s^h = 7 \cdot 27 = 189$ which is pretty much calculation per pixel. The separable implementation needs only $F_s = F_s^v + F_s^h = 7 + 27 = 34$ which is much reduction.

In the next experiment, we show the option to handle different degradation types by application large filter sizes in different situations. In Figure 6, we simulated motion blur of 25 pixels in a direction of 30 degrees and estimated the gain of quality for different horizontal and vertical filter sizes.

It becomes obvious that filter sizes in the range of blur are required to obtain good performance of the restoration. Due to the angle of motion the horizontal component is larger than the vertical component. Correspondingly the the saturation of gain value is reached quicker.

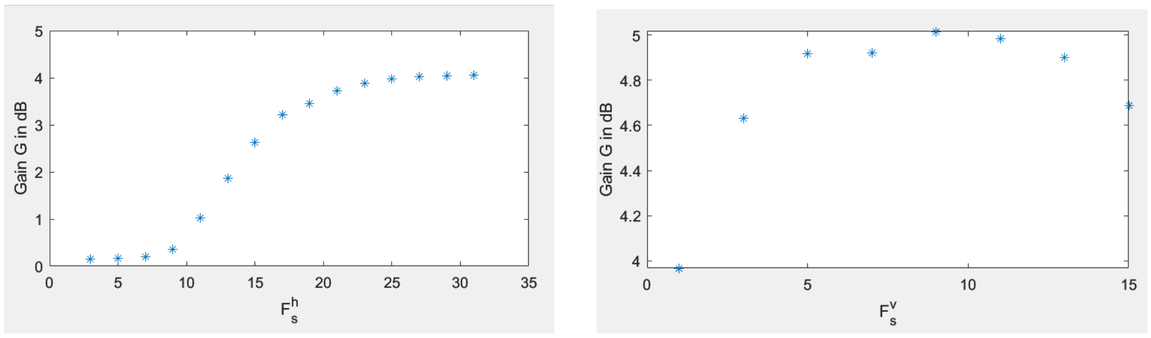
In 7, we additionally superimposed normally distributed noise on a motion blurred image by 25 pixels in horizontal direction and estimated the quality gain for different horizontal filters of size F_s^h. The graph shows that small filter sizes lead to weak gain, especially for higher noise levels.

In 8, we additionally superimposed normally distributed noise on the same motion blurred image and estimated the quality gain for different 2-dimensional filters of same horizontal and vertical sizes. The graph shows that small filter sizes lead to weak gain, especially for higher noise levels.

All the above experiments can be performed in real-time on the Edge TPU for typical video streams because the filter sizes are below of the estimated limits calculated in section 4.1 which is in the range of one hundered.

7 CONCLUSIONS

We showed how a Deep Learning model for a deconvolution filter can be dimensioned from a known input output relation of an image capturing system and how it can be inferred on an Edge TPU. The filter has been estimated without using knowledge of degrading filter but on the basis of known original and degraded image. This implicitly considers the noise impact and



(a) (b)

Figure 5: Dependency of quality gain G on filter sizes F_s^h and F_s^v for the inference example: 5a dependency of Gain on horizontal filter size F_s^h for motion blurred image $\vec{g}^b$; 5b dependency of Gain on vertical filter size F_s^v with additional noise superposition by $PSNR = 40\,\text{dB}$.

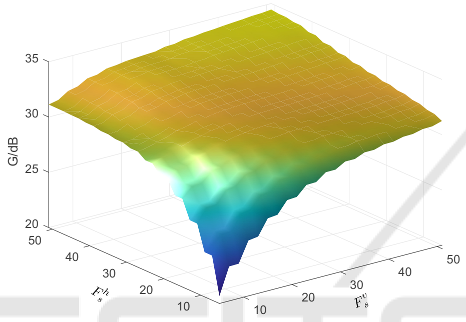


Figure 6: Gain G for different horizontal and vertical filter sizes with a motion blur of 25 pixels in direction 30 degrees.

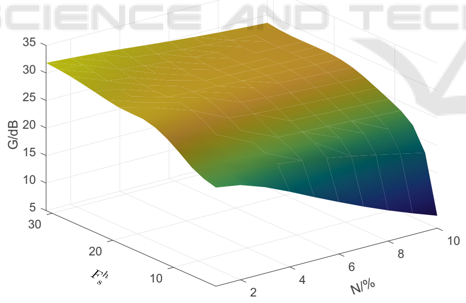


Figure 7: Gain G for horizontal motion and different horizontal filter sizes F_s^h and noise levels.

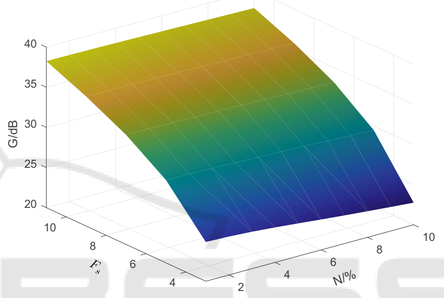


Figure 8: Gain G for different horizontal filter sizes $F_s = F_s^v = F_s^h$ and noise levels.

avoids the otherwise necessity of inverting the degrading systems.

The Edge TPU allows very long filter sizes to be calculated in real-time for low-level processing of video streams. As part of the digittal camera pipeline it can act as an electronic lens when the large filter is not only used for noise reduction and deblurring but also for the removal of larger motion blur or geometric lens distortions. We showed promising results for non-separable filter sizes in the range of up to one hundred. With the potentially possible separable filter size of about two thousand interesting applications could be possible which is part for future work.

The Tensor Flow provided function TransposeConv also has potentially interesting applications for image error compensation that are worth exploring in further work. Unfortunately, the potentially large filter sizes due to the large computational power cannot be implemented in a location-variant manner, severely limiting its applicability. For such applications, tailored hardware solutions therefore still seem to be the means of choice.

REFERENCES

Abeykoon, V., Liu, Z., Kettimuthu, R., Fox, G., and Foster, I. (2019). Scientific Image Restoration Anywhere. In *2019 IEEE/ACM 1st Annual Workshop on Large-scale Experiment-in-the-Loop Computing (XLOOP)*, pages 8–13, Denver, CO, USA. IEEE.

Andrews, H. C. and Hunt, B. R. (1977). *Digital image restoration*. Prentice-Hall signal processing series. Prentice-Hall, Englewood Cliffs, N.J.

Basler (2021a). Design an Eye-Catching Vision System for Machine Learning with the i.MX 8M Plus Applications Processor. embedded_vision/nxp-2021-1/NXP-TECH-SESSION-EYE-CATCHING-VISION-SYSTEM-MLAI.pdf.

Basler (2021b). The NXP i.MX 8M Plus Applications Processor: Bringing High-Performance Machine Learning to the Edge. embedded_vision/nxp-2021-2/AI-TO-THE-EDGE-WP-1-1.pdf.

Civit-Masot, J., Luna-Perejón, F., Corral, J. M. R., Domínguez-Morales, M., Morgado-Estévez, A., and Civit, A. (2021). A study on the use of Edge TPUs for eye fundus image segmentation. *Engineering Applications of Artificial Intelligence*, 104:104384.

Coral (2021a). Models.

Coral (2021b). TensorFlow models on the Edge TPU. TPU_Preprocessing/coral_tensorflow_nodate1/models-intro.html.

Coral (2022). USB Accelerator. https://coral.ai/products/accelerator.

Google (2021). TensorFlow Lite | ML for Mobile and Edge Devices.

Harald Bögeholz (2017). Künstliche Intelligenz: Architektur und Performance von Googles KI-Chip TPU. TPU_Preprocessing/online_kunstliche_nodate_heise2/Kuenstliche-Intelligenz-Architektur-und-Performance-von-Googles-KI-Chip-TPU-3676312.html.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. TPU_Preprocessing/LeCun1998GradientbasedLA/Convolution_nets.pdf.

Markovtsev, V. (2019). Hacking Google Coral Edge TPU: motion blur and Lanczos resize.

Merenda, M., Porcaro, C., and Iero, D. (2020). Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors (Basel, Switzerland)*, 20(9):2533. tpu-paper/merenda_edge_2020/Merendaetal.-2020-EdgeMachineLearningforAI-EnabledIoTDevices.pdf.

Sun, Y. and Kist, A. M. (2022). Deep Learning on Edge TPUs. arXiv:2108.13732 [cs].

Wikipedia contributors (2021). Tensor processing unit — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Tensor_Processing_Unit&oldid=1056864312. [Online; accessed 20-January-2022].

Yazdanbakhsh, A., Seshadri, K., Akin, B., Laudon, J., and Narayanaswami, R. (2021). An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. *arXiv:2102.10423 [cs]*. arXiv: 2102.10423 tpu/yazdanbakhsh_evaluation_2021/2102.10423.pdf.

Zeiler, M., Krishnan, D., Taylor, G., and Fergus, R. (2010). Deconvolutional networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535. TPU_Preprocessing/Zeiler-2010-1/deconvolutionalnetworks.pdf.