BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs

Valentin Bazarevsky Yury Kartynnik Andrey Vakunov Karthik Raveendran Matthias Grundmann
Google Research
1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA
{valik, kartynnik, vakunov, krav, grundman}@google.com

Abstract

We present BlazeFace, a lightweight and well-performing face detector tailored for mobile GPU inference. It runs at a speed of 200–1000+ FPS on flagship devices. This super-realtime performance enables it to be applied to any augmented reality pipeline that requires an accurate facial region of interest as an input for task-specific models, such as 2D/3D facial keypoint or geometry estimation, facial features or expression classification, and face region segmentation. Our contributions include a lightweight feature extraction network inspired by, but distinct from MobileNetV1/V2, a GPU-friendly anchor scheme modified from Single Shot MultiBox Detector (SSD), and an improved tie resolution strategy alternative to non-maximum suppression.

1. Introduction

In recent years, a variety of architectural improvements in deep networks ([4, 6, 8]) have enabled real-time object detection. In mobile applications, this is usually the first step in a video processing pipeline, and is followed by task-specific components such as segmentation, tracking, or geometry inference. Therefore, it is imperative that the object detection model inference runs as fast as possible, preferably with the performance much higher than just the standard real-time benchmark.

We propose a new face detection framework called BlazeFace that is optimized for inference on mobile GPUs, adapted from the Single Shot Multibox Detector (SSD) framework [4]. Our main contributions are:

1. Related to the inference speed:
   1.1. A very compact feature extractor convolutional neural network related in structure to MobileNetV1/V2 [3, 9], designed specifically for lightweight object detection.
   1.2. A novel GPU-friendly anchor scheme modified from SSD [4], aimed at effective GPU utilization. *Anchors* [8], or *priors* in SSD terminology, are predefined static bounding boxes that serve as the basis for the adjustment by network predictions and determine the prediction granularity.
2. Related to the prediction quality: A tie resolution strategy alternative to non-maximum suppression [4, 6, 8] that achieves stabler, smoother tie resolution between overlapping predictions.

2. Face detection for AR pipelines

While the proposed framework is applicable to a variety of object detection tasks, in this paper we focus on detecting faces in a mobile phone camera viewfinder. We build separate models for the front-facing and rear-facing cameras owing to the different focal lengths and typical captured object sizes.

In addition to predicting axis-aligned face rectangles, our BlazeFace model produces 6 facial keypoint coordinates (for eye centers, ear tragions, mouth center, and nose tip) that allow us to estimate face rotation (roll angle). This enables passing a rotated face rectangle to later task-specific stages of the video processing pipeline, alleviating the requirement of significant translation and rotation invariance in subsequent processing steps (see Section 5).

3. Model architecture and design

BlazeFace model architecture is built around four important design considerations discussed below.

Enlarging the receptive field sizes. While most of the modern convolutional neural network architectures (including both MobileNet [3, 9] versions) tend to favor 3×3 convolution kernels everywhere along the model graph, we note that the depthwise separable convolution computations are dominated by their pointwise parts. On an $s \times s \times c$ input tensor, a $k \times k$ depthwise convolution involves s^2ck^2 multiply-add operations, while the subsequent 1×1 convolution into d output channels is comprised of s^2cd such operations, within a factor of d/k^2 of the depthwise part.

In practice, for instance, on an Apple iPhone X with the Metal Performance Shaders implementation [1], a 3×3

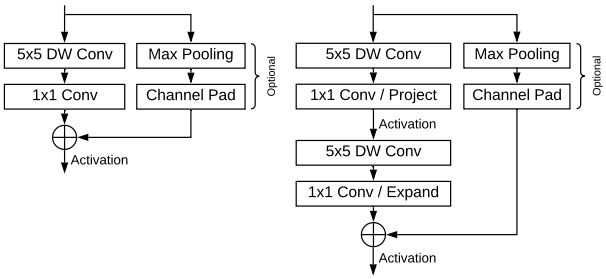

Figure 1. BlazeBlock (left) and double BlazeBlock

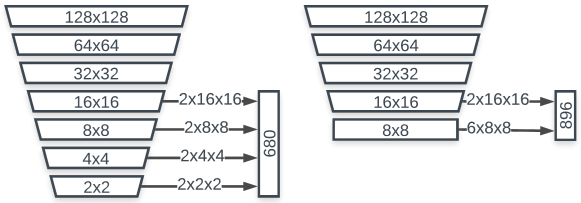

Figure 2. Anchor computation: SSD (left) vs. BlazeFace

depthwise convolution in 16-bit floating point arithmetic takes 0.07 ms for a 56×56×128 tensor, while the subsequent 1×1 convolution from 128 to 128 channels is 4.3× slower at 0.3 ms (this is not as significant as the pure arithmetic operation count difference due to fixed costs and memory access factors).

This observation implies that increasing the kernel size of the depthwise part is relatively cheap. We employ 5×5 kernels in our model architecture bottlenecks, trading the kernel size increase for the decrease in the total amount of such bottlenecks required to reach a particular receptive field size (Figure 1).

A MobileNetV2 [9] bottleneck contains subsequent depth-increasing *expansion* and depth-decreasing *projection* pointwise convolutions separated by a non-linearity. To accommodate for the fewer number of channels in the intermediate tensors, we swap these stages so that the residual connections in our bottlenecks operate in the "expanded" (increased) channel resolution.

Finally, the low overhead of a depthwise convolution allows us to introduce another such layer between these two pointwise convolutions, accelerating the receptive field size progression even further. This forms the essence of a *double BlazeBlock* that is used as the bottleneck of choice for the higher abstraction level layers of BlazeFace (see Figure 1, right).

Feature extractor. For a specific example, we focus on the feature extractor for the front-facing camera model. It has to account for a smaller range of object scales and therefore has lower computational demands. The extractor takes an RGB input of 128×128 pixels and consists of a 2D convolution followed by 5 single BlazeBlocks and 6 double BlazeBlocks (see Table 4 in Appendix A for the full layout). The highest tensor depth (channel resolution) is 96, while the lowest spatial resolution is 8×8 (in contrast to SSD, which reduces the resolution all the way down to 1×1).

Anchor scheme. SSD-like object detection models rely on pre-defined fixed-size base bounding boxes called *priors*, or *anchors* in Faster-R-CNN [8] terminology. A set of regression (and possibly classification) parameters such as center offset and dimension adjustments is predicted for each anchor. They are used to adjust the pre-defined anchor position into a tight bounding rectangle.

It is a common practice to define anchors at multiple resolution levels in accordance with the object scale ranges. Aggressive downsampling is also a means for computational resource optimization. A typical SSD model uses predictions from 1×1, 2×2, 4×4, 8×8, and 16×16 feature map sizes. However, the success of the Pooling Pyramid Network (PPN) architecture [7] implies that additional computations could be redundant after reaching a certain feature map resolution.

A key feature specific to GPU as opposed to CPU computation is a noticeable fixed cost of dispatching a particular layer computation, which becomes relatively significant for deep low-resolution layers inherent to popular CPU-tailored architectures. As an example, in one experiment we observed that out of 4.9 ms of MobileNetV1 inference time only 3.9 ms were spent in actual GPU shader computation.

Taking this into consideration, we have adopted an alternative anchor scheme that stops at the 8×8 feature map dimensions without further downsampling (Figure 2). We have replaced 2 anchors per pixel in each of the 8×8, 4×4 and 2×2 resolutions by 6 anchors at 8×8. Due to the limited variance in human face aspect ratios, limiting the anchors to the 1:1 aspect ratio was found sufficient for accurate face detection.

Post-processing. As our feature extractor is not reducing the resolution below 8×8, the number of anchors overlapping a given object significantly increases with the object size. In a typical non-maximum suppression scenario, only one of the anchors "wins" and is used as the final algorithm outcome. When such a model is applied to subsequent video frames, the predictions tend to fluctuate between different anchors and exhibit temporal jitter (human-perceptible noise).

To minimize this phenomenon, we replace the *suppression* algorithm with a *blending* strategy that estimates the regression parameters of a bounding box as a weighted mean between the overlapping predictions. It incurs virtually no additional cost to the original NMS algorithm. For our face

detection task, this adjustment resulted in a 10% increase in accuracy.

We quantify the amount of jitter by passing several slightly offset versions of the same input image into the network and observing how the model outcomes (adjusted to account for the translation) are affected. After the described tie resolution strategy modification, the jitter metric, defined as the root mean squared difference between the predictions for the original and displaced inputs, dropped down by 40% on our frontal camera dataset and by 30% on a rear-facing camera dataset containing smaller faces.

4. Experiments

We trained our model on a dataset of 66K images. For evaluation, we used a private geographically diverse dataset consisting of 2K images. For the frontal camera model, only faces that occupy more than 20% of the image area were considered due to the intended use case (the threshold for the rear-facing camera model was 5%).

The regression parameter errors were normalized by the inter-ocular distance (IOD) for scale invariance, and the median absolute error was measured to be 7.4% of IOD. The jitter metric evaluated by the procedure mentioned above was 3% of IOD.

Table 1 shows the average precision (AP) metric [5] (with a standard 0.5 intersection-over-union bounding box match threshold) and the mobile GPU inference time for the proposed frontal face detection network and compares it to a MobileNetV2-based object detector with the same anchor coding scheme (MobileNetV2-SSD). We have used TensorFlow Lite GPU [2] in 16-bit floating point mode as the framework for inference time evaluation.

Model	Average Precision	Inference Time, ms (iPhone XS)
MobileNetV2-SSD	97.95%	2.1
Ours	**98.61%**	**0.6**

Table 1. Frontal camera face detection performance

Table 2 gives a perspective on the GPU inference speed for the two network models across more flagship devices.

Device	MobileNetV2-SSD, ms	Ours, ms
Apple iPhone 7	4.2	**1.8**
Apple iPhone XS	2.1	**0.6**
Google Pixel 3	7.2	**3.4**
Huawei P20	21.3	**5.8**
Samsung Galaxy S9+ (SM-G965U1)	7.2	**3.7**

Table 2. Inference speed across several mobile devices

Table 3 shows the amount of degradation in the regression parameter prediction quality that is caused by the smaller model size. As explored in the following section, this does not necessarily incur a proportional degradation of the whole AR pipeline quality.

Model	Regression error	Jitter metric
MobileNetV2-SSD	**7.4%**	**3.6%**
Ours	10.4%	5.3%

Table 3. Regression parameters prediction quality

5. Applications

The proposed model, operating on the full image or a video frame, can serve as the first step of virtually any face-related computer vision application, such as 2D/3D facial keypoints, contour, or surface geometry estimation, facial features or expression classification, and face region segmentation. The subsequent task in the computer vision pipeline can thus be defined in terms of a proper facial crop. Combined with the few facial keypoint estimates provided by BlazeFace, this crop can be also rotated so that the face inside is centered, scale-normalized and has a roll angle close to zero. This removes the requirement of significant translation and rotation invariance from the task-specific model, allowing for better computational resource allocation.

We illustrate this pipelined approach with a specific example of face contour estimation. In Figure 3, we show how the output of BlazeFace, i.e. the predicted bounding box and the 6 keypoints of the face (red), are further refined by a more complex face contour estimation model applied to a slightly expanded crop. The detailed keypoints yield a finer bounding box estimate (green) that can be reused for tracking in the subsequent frame without running the face detector. To detect failures of this computation saving strategy, the contours model can also detect whether the face is indeed present and reasonably aligned in the provided rectangular crop. Whenever that condition is violated, the BlazeFace face detector is run on the whole video frame again.

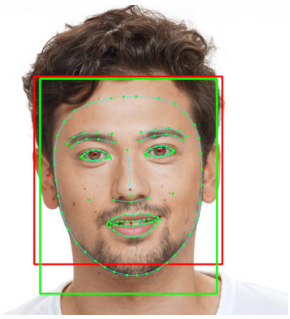

Figure 3. Pipeline example (best viewed in color). Red: BlazeFace output. Green: Task-specific model output.

The technology described in this paper is driving major AR self-expression applications and AR developer APIs on mobile phones.

References

[1] Metal performance shaders. https://developer.apple.com/documentation/metalperformanceshaders. [Online; accessed April 19, 2019]. 1

[2] TFLite on GPU. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/delegates/gpu. [Online; accessed April 19, 2019]. 3

[3] Andrew Howard *et al*. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[4] Wei Liu *et al*. SSD: Single shot MultiBox detector. In *European conference on computer vision*, pages 21–37, 2016. 1

[5] Mark Everingham, Luc Gool, Christopher K. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vision*, 88(2):303–338, 2010. 3

[6] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1

[7] Pengchong Jin, Vivek Rathod, and Xiangxin Zhu. Pooling pyramid network for object detection. *arXiv preprint arXiv:1807.03284*, 2018. 2

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1, 2

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 1, 2

Appendix A. Feature extraction network architecture

Layer/block	Input size	Conv. kernel sizes
Convolution	128×128×3	128×128×3×24
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 1×1×24×24
Single BlazeBlock	64×64×24	5×5×24×1 (stride 2) 1×1×24×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Single BlazeBlock	32×32×48	5×5×48×1 1×1×48×48
Double BlazeBlock	32×32×48	5×5×48×1 (stride 2) 1×1×48×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	16×16×96	5×5×96×1 (stride 2) 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96
Double BlazeBlock	8×8×96	5×5×96×1 1×1×96×24 5×5×24×1 1×1×24×96

Table 4. BlazeFace feature extraction network architecture