EDGE MACHINE LEARNING FOR FACE DETECTION

Geffen Cooper, B.S. Manjunath, and Yogananda Isukapalli
University of California, Santa Barbara
Santa Barbara, CA 93106
geffen_cooper@ucsb.edu, manj@ucsb.edu, yoga@ucsb.edu

ABSTRACT

This paper describes an implementation of edge machine learning for vision-based classification and detection tasks. In edge machine learning, machine and deep learning algorithms are executed locally on embedded devices rather than on more powerful computers or the cloud. The main task explored is face detection using a low-power microcontroller. This device utilizes a convolutional neural network (CNN) accelerator that optimizes convolution and pooling operations for fast power-efficient inference. Development for this system requires building and training a hardware-limited CNN rather than fine-tuning a pre-trained state-of-the-art model. The development process is discussed along with the constraints of this embedded device.

INTRODUCTION

Convolutional Neural Networks (CNNs) have become ubiquitous in computer vision as they have been very successful when applied to object classification and detection tasks. However, one of the main limitations of deep-learning based solutions in computer vision is the computational complexity of neural networks. State-of-the-art classification and detection networks often contain millions of parameters that require powerful compute resources to train [1]. Even after training, a network can require hundreds of megabytes of storage and intensive computation for each forward pass. Accordingly, these deep-learning solutions may sometimes only be executed on remote powerful computers with the aid of GPUs. However, this paradigm where data is processed on a secondary machine may not be feasible or desirable if the application has real-time requirements or power constraints such as on standalone embedded devices. As a result, the intersection of machine learning and embedded systems has become very active especially with the explosion in the number of embedded devices. While current applications are still limited, edge machine learning will enable embedded devices to process data from sensors locally in real-time. This can be applied to standalone medical devices for monitoring patients or augment everyday appliances with speech and facial recognition.

While training a network still requires powerful compute resources, it is possible to do inference on an embedded system. This paper explores a specific implementation of edge machine learning for vision-based classification and detection tasks that uses a low-power microcontroller with a CNN inference accelerator called the MAX78000 [2]. While this specialized hardware makes

deep learning at the edge possible, it understandably comes with limitations. These include approximately 442 kilobytes for storage of the network's weights, 512 kilobytes of data memory for intermediate processing during a forward pass, quantization of network parameters to 8-bit values, and specific limitations on the network architecture such as kernel size, stride size, and input dimensions. The complete hardware specifications are discussed in [2] and [3]. These constraints make the development of deep learning models more challenging. A common practice in development of deep learning models is transfer learning in which a pre-trained model is extended or trained further for the desired task. Given the previously stated limitations, using a pre-trained state-of-the-art model is not possible and networks must be built to fit the hardware constraints of the microcontroller. In addition to hardware constraints, bare-metal mirocontrollers such as these do not have an operating system that can standardize and simplify the development process of deep learning models. As such, additional processing steps are required at inference time such as extracting pixel data from the on-board camera module.

The rest of the paper explains the four classification and detection tasks explored: handwritten digit classification, detecting if a face is in an image (binary classification), localizing a face in an image with a bounding box, and localizing eyes in an image with dots. Due to time constraints the task of face detection was limited to the simple case where a single face is present and there is good lighting. The network architecture built for face detection is based on VGG16 but only requires approximately 150 kilobytes of memory and has an inference time of approximately 4 milliseconds which makes it suitable for real-time applications. The remainder of the paper discusses the details and steps taken to accomplish the classification and detection tasks.

CLASSIFYING HANDWRITTEN DIGITS

The first task explored was classifying handwritten digits using a model trained on the MNIST dataset [4]. While a very simple CNN architecture can achieve better than 95% accuracy on the MNIST test set, this task uses live data that comes from the on-board camera module during inference-time. Accordingly, additional preprocessing steps are required in order to classify digits using real-time camera data. These steps and the program control flow are outlined in detail in Figure 1. The camera data must be converted into the same format as the training data before being fed into the CNN for inference.

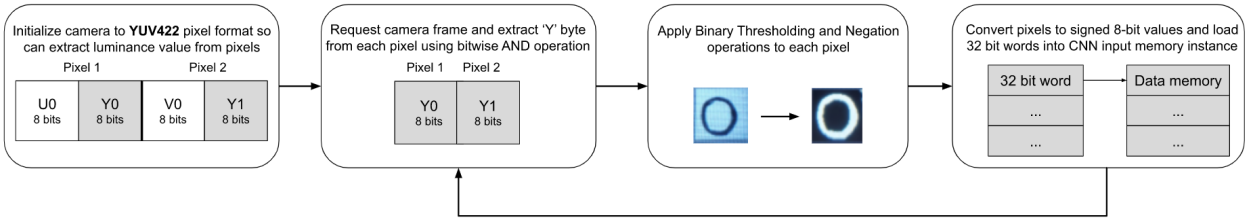


Figure 1: Preprocessing Steps and Control Flow for Real-Time Handwritten Digit Classification.

The model used for classifying the digits is shown in Figure 2. The 8-bit quantized version of this model achieves close to 99% accuracy on the MNIST test set. The parameters take up 22

kilobytes of memory and each forward pass takes less than 2 milliseconds. The training process for this microcontroller requires the use of the Maxim Integrated software tools which provides a modified version of PyTorch that augments the standard modules with information about the hardware (clipping, rounding, fusing layers) so that a model can be correctly synthesized and loaded onto the accelerator. Detailed information about this process can be found in [3].

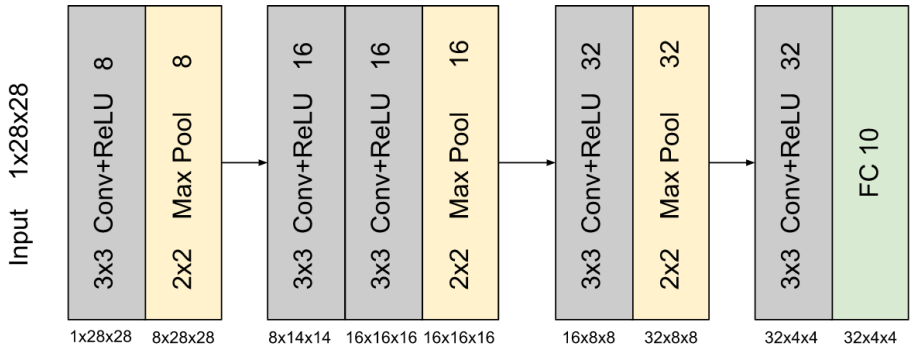


Figure 2: CNN Architecture.

BUILDING A BINARY CLASSIFIER

The second task explored was building a binary classifier to determine if a single face is present in an image. This first involved assembling a dataset of images with a single face present and no face present. The datasets used are summarized in Table 1. The total number of images is approximately 140 thousand with 10% of the images from each category being reserved for the test set. All images were resized to 80x80 and converted to grayscale.

Dataset	Description	# of Images	Sample Image
FFHQ [5]	Single Faces	70K	
Caltech 256 [6]	Random Objects	30K	
MIT Indoor Scenes [7]	Indoor Scenes	15K	
Stanford STL-10 [8]	Random Objects	25K used of 100K	

Table 1: Dataset Summary.

The architecture used for the binary classifier is based off of VGG16 [9]. VGG16 is a well known architecture used for image classification and while it has many parameters it is very simple. The

original VGG16 architecture and the modified version, labeled as Mini VGG, are shown in Figure 3. The characteristics from VGG16 that Mini VGG tries to emulate are doubling the channel dimension after pooling, putting multiple convolution layers after pooling, and using multiple fully connected layers at the output. While the two architectures are not exactly the same, the Mini VGG architecture heuristically performs well and classifies approximately 98% of the images in the test set correctly. In addition, the hardware cannot flatten 2D layers with more than 1,024 parameters into fully connected layers so several pooling layers must be used to reduce the parameters. Figure 4 shows two sample frames being classified in real-time as either containing or not containing a face.

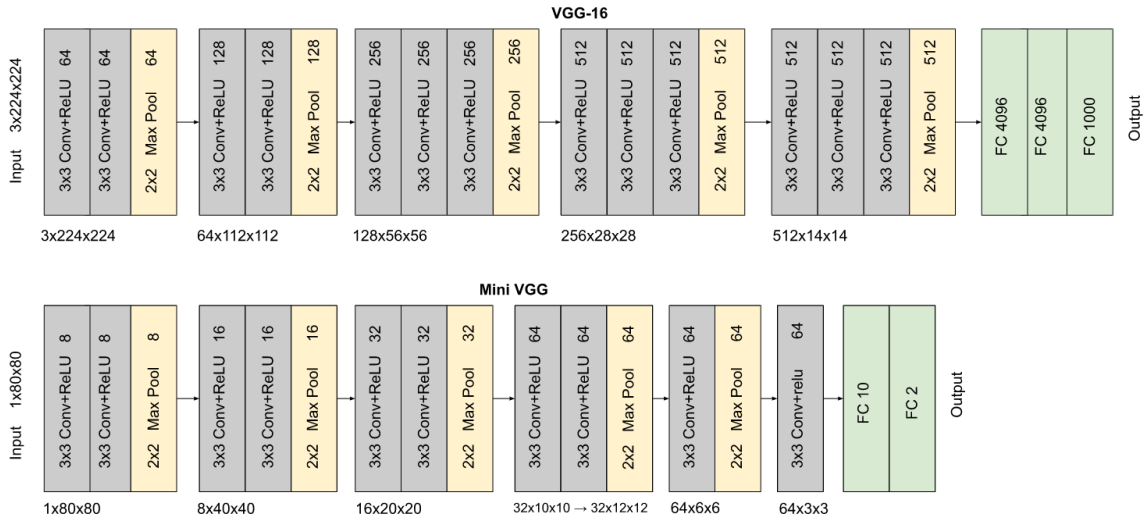
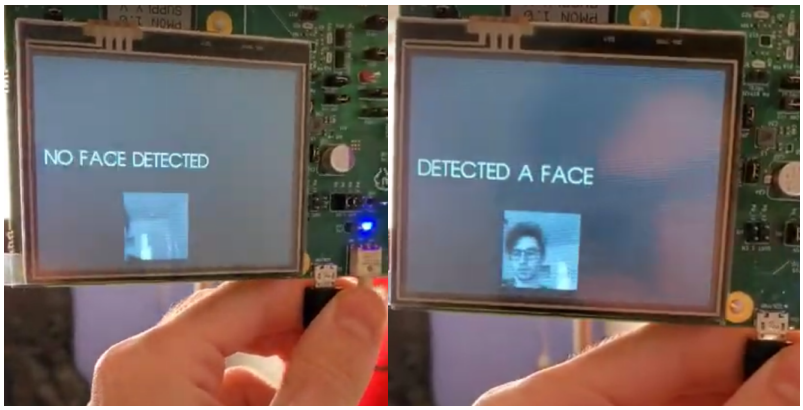


Figure 3: Binary Classifier CNN Architecture.



(a) No Face Present. (b) Face Present.

Figure 4: Binary Classifier Live Output.

The faces in the FFHQ dataset are all centered and take up the majority of the image. This lack of variation resulted in poor performance when the network was tested with real-time camera data since a face may be small or in the corner. Therefore, translation, scaling, and rotation data

augmentations were used in order to make the network more robust to different conditions. While this significantly improved the performance, the network still has difficulty in non ideal lighting conditions because the training data has very good contrast whereas the camera data may not.

USING THE BINARY CLASSIFIER AS A FEATURE EXTRACTOR

A common practice in deep-learning is to utilize pre-trained models rather than building and training a network from scratch [10]. Similarly, the convolution layers of the binary classifier network discussed in the last section were used as the backbone of the face detection network. Since the binary classifier network was already trained on a dataset of faces, the output of the last convolution layer (prior to classification) can be thought of as high level facial features. This methodology assumes the binary classifier is searching for or is activated by facial features. To visualize what the network is activated by, a tool called SHAP (SHapley Additive exPlanations) was used [11]. This tool can help identify how much each portion of an input image is contributing to a given output class. Figure 5 shows the output of the SHAP tool for three sample images. In the images, the red (darker) pixels are the regions that increase the model's output for the face class.

Figure 5: SHAP Output.

From these three examples, and many others, the network seems to recognize prominent facial features such as the eyes. Based on these heuristic results, the binary classifier network was extended as shown in Figure 6. This network is identical to the binary classifier with the addition of a fully connected layer for bounding box outputs. These are the top left corner and the width and height of the box. In Figure 6, the grey (darker) boxes represent layers where the parameters are frozen and do not update during training (backpropagation). The green (lighter) box, denoted as FC 4, represents a layer where the weights are being updated during training. Overall, the convolution layers are being used as a feature extractor and only one output layer is being trained to predict the face location using a bounding box.

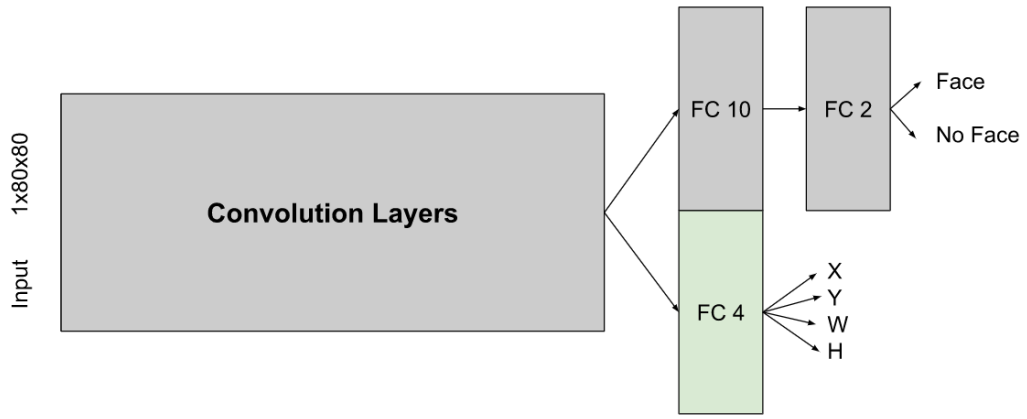


Figure 6: Face Detector Architecture.

Bounding box detection differs from binary classification in that it is a regression task rather than a classification task. One of the main differences when training is the loss function since the output now represents a continuous set of values rather than a finite set of discrete classes. Common loss functions for regression tasks are based on the L1 and L2 norm. These are shown in (1) and (2) for a given prediction $\hat{Y}$ and corresponding label Y.

$$L1 = \sum_{i=1}^{n} |Y_i - \hat{Y}_i| \tag{1}$$

$$L2 = \sqrt{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2} \tag{2}$$

In addition, a common evaluation metric for bounding box detection is IoU (intersection over union) shown in Figure 7.
The IoU is 0 when there is no intersection and 1 when there is complete overlap of the predicted box and the target box. [12] shows that minimizing an L1 or L2 based loss does not necessarily cause the IoU to improve because two predicted boxes may have the same L1 or L2 loss but different IoU values. Instead, athey propose an IoU based loss metric called generalized IoU which is shown in (3) and (4) where A and B are the predicted and labeled bounding boxes and C is the smallest region that includes both (When there is overlap, $GIoU = IoU$). C accounts for the case when there is no overlap.

$$GIoU = IoU - \frac{|C \setminus (A \cup B)|}{|C|} \tag{3}$$

$$Loss = 1 - GIoU \tag{4}$$

While (4) was not directly used for the loss function, a similar IOU based loss was used rather than an L1 or L2 based loss. The loss function used for the face detector is shown in (5) where a small constant is added to the overlap area in case there is no overlap.

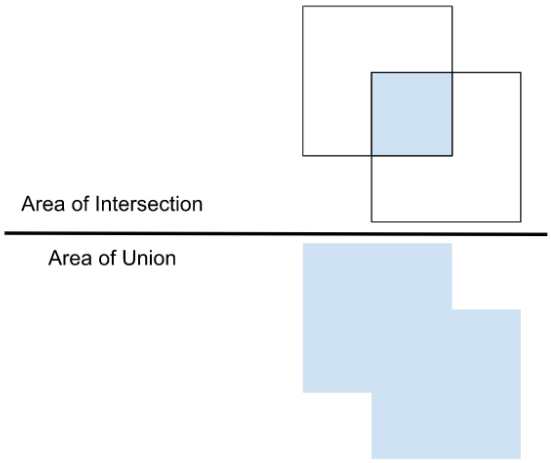
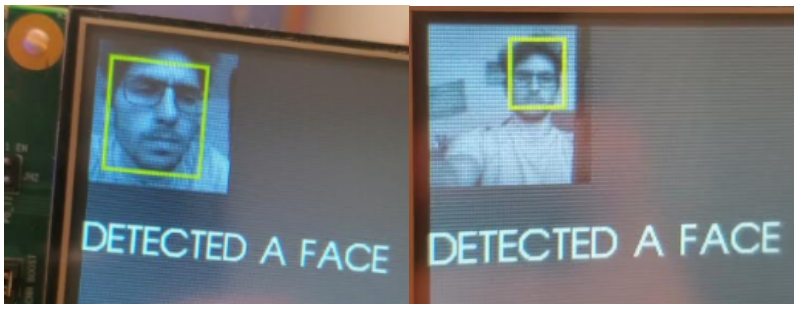


Figure 7: Intersection Over Union (IoU)

$$Loss = -\ln(IOU) \tag{5}$$

The dataset used for training the face detector was a cropped subset of the WIDER Face dataset [13] that contained images of a single face. This produced roughly 1.3k images with 10% being used for the test set. Figure 8 shows two frames of a face being localized in real-time from a closer and farther distance.



(a) Face Close. (b) Face Far.

Figure 8: Face Detector Live Output.

The final task explored was localization of the eyes. For this task, the same architecture was used (as shown in Figure 6). However, the output vector of the network for localizing eyes is the x and y coordinates of the left and right eyes, (x_L, y_L, x_R, y_R), rather than the bounding box parameters, (x, y, w, h). For this task the $L2$ norm was used for each coordinate as the loss because the output is a set of two points rather than a box so the evaluation metric should be the euclidean distance. Given the limited amount of time and training data, this task had limited success. While the eyes were able to be tracked under certain conditions, the network was very sensitive to facial pose and

lighting which is expected given that the training data did not have enough variance for robust eye tracking. Figure 9 shows two frames of eyes being localized in real-time.

(a) Eyes Localized. (b) Eyes Incorrectly Localized.

Figure 9: Eye Detector Live Output.

CONCLUSIONS

The observed results reflect how reasonably complex vision based tasks such as object detection are achievable on a low-power microcontroller with a CNN accelerator. While the performance is not robust to more complex settings with multiple faces or poor lighting, this project mainly serves as a proof of concept rather than an end product. Applications of such a system could be used for driver drowsiness detection. The final model used approximately 150 kilobytes out of a maximum of approximately 442 kilobyte weight memory so there is a potential for building deeper and more complex architectures with more parameters and a larger input image resolution. For a more complete analysis, deeper or more complex network architectures should be explored, different input image resolutions should be tested, and more representative training data should be used to account for the imperfections of live camera data.

ACKNOWLEDGMENTS

We would like to thank Brian Rush and Robert Muchsel from Maxim Integrated for their technical assistance in working with the MAX78000.

REFERENCES

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review*, vol. 53, pp. 5455–5516, Dec 2020.

[2] "Max78000 artificial intelligence microcontroller with ultra-low-power convolutional neural network accelerator - maxim integrated." https://www.maximintegrated.com/en/products/microcontrollers/MAX78000.html.

[3] "Maxim integrated ai github repository." https://github.com/MaximIntegratedAI.

[4] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[5] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019.

[6] G. Griffin, A. Holub, and P. Perona, "The Caltech 256. Caltech Technical Report.," 2006.

[7] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 413–420, 2009.

[8] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 215–223, PMLR, 11–13 Apr 2011.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[10] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," 2014.

[11] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4765–4774, Curran Associates, Inc., 2017.

[12] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," 2019.

[13] S. Yang, P. Luo, C. C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.