Semantic Segmentation using the EdgeTPU to Assist Visually Impaired People Navigate Indoor Areas

Victor Tran
Department of Computer Science
California State University, Fullerton
Fullerton, United States of America
victorvantran@csu.fullerton.edu

Aneesh Reddy Sannapu
Department of Computer Science
California State University, Fullerton
Fullerton, United States of America
aneeshreddysannapu@csu.fullerton.edu

Kayhan Bakian
Department of Computer Science
California State University, Fullerton
Fullerton, United States of America
kayhanbakian@csu.fullerton.edu

Kanika Sood
Department of Computer Science
California State University, Fullerton
Fullerton, United States of America
kasood@fullerton.edu

***Abstract*—Recognizing the limited supply and high cost of alternative solutions such as guides, it's incredibly important to create an affordable and accessible means of mapping walkable paths for visually impaired individuals. In this paper, we present a model built around image segmentation and object detection that utilizes a mix of deep learning strategies and traditional models to identify obstacles in real-time. Our approach lays the foundation for changing the way the visually impaired can sense the world by offering an assistance system for indoor navigation.**

***Index Terms*—semantic segmentation, EdgeTPU, DeepLabV3, Residual Network, Post-quantize-aware training, Full Integer Quantization.**

I. Introduction

With technological advancements ushering in a new age of progress, new ways to ameliorate those living with the symptoms of disabilities have been pursued. The ailment we focused on in this paper is that of visual impairment. Currently, one of the most common tools utilized by the visually impaired is the white cane, which helps them gather a sense of their surroundings. However, only 2-8% of individuals use them [1]. The majority of individuals rely on their usable vision, a seeing guide, or a guide dog [2]. While seeing-eye dogs and caregivers are very useful, they are not readily available or affordable in some cases.

Semantic segmentation is based on the idea of breaking up a single image into multiple segments that can be analyzed, allowing us to take a very complex initial image and simplifying it to allow for the model to perform better. The concept has been tinkered with since the 1980s but with the advancement of deep learning and neural networks, it's become an incredibly powerful tool with applications in medical analysis, image compression, video surveillance, and even augmented reality among others [3].

II. Related Work

Previous approaches to utilizing image recognition software for object detection has been thwarted by the depth of complexity with which it must be able to analyze. The combination of low-level images and high amounts of context necessary in order for the program to recognize an image in real-time made it so the performance of traditional machine learning models tended to stagnate. However, with the onset of Deep Learning algorithms and the evolution of Convolutional Neural Networks, the effectiveness of Object Detection has greatly increased [4].

This work is in line with the technological landscape as it puts a focus on real-time detection by mixing neural networks and traditional machine learning models to identify the best performance which can trace the walkable path for an individual who is visually impaired. This project builds on the adjusted focus of using sensory outputs to map the path around an individual. While previous papers used audio cues, in the future we will attempt to add a heat map that gets warmer the closer to an object an individual is [5].

After analyzing previous works on the topic of object detection, the most popular method noted was the YOLO algorithm. The algorithm divides the whole image into N*N equal-size grids, where each cell will predict if there is an object or not by assigning 0 if no object is detected and greater than 0, i.e. (0.1 - 1) if there is an object. The bounding boxes are formed by analyzing each cell and its corresponding probability. Intersection over unions is then used to combine all the bounding boxes constructed for each individual object, detecting the object.[6]

Previous studies of walkable path detection have used tensor graphs. The tensor graph is positioned in the center of the image and is subdivided into three parts: left boundary, middle

boundary, and right boundary. If the detected object falls under the middle bounding box, it alerts the person to stop walking. it then checks if either the left or right boundary are obstacle free. If an accessible path is found, it suggests the person to move towards that side. The whole process is repeated until the person reaches their destination [7].

III. Problem Statement and Approach

There are currently more than 2.2 billion individuals with near or distance vision impairment, and over 43 million suffering from blindness [1]. One tool that can be explored is the usage of live camera feeds and machine learning algorithms to help individuals detect obstacles in their path, ensuring they can avoid collisions. This would be easily accessible and could be an attachment to a regular cane in the future. The camera feed would translate images into arrays of pixels which would be processed using semantic segmentation and objection detection.

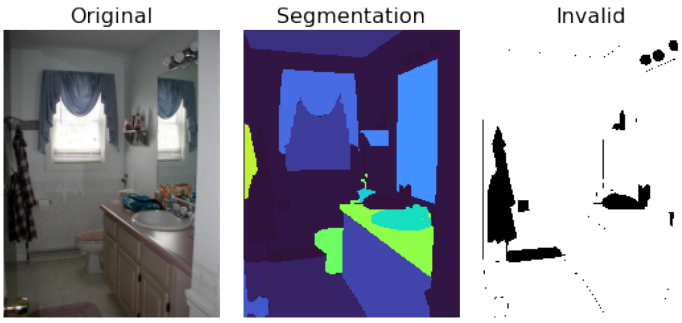
This work utilizes the Google Coral with the Edge Tensor Processing Unit (EdgeTPU) chip which allows us to run specially trained, small, neural network models on portable devices with fast inferences times [8]. Recognizing that constant environment shifts would necessitate quick processing, by using the EdgeTPU we can get around 10 segmentation map updates per second. The EdgeTPU also allows us to use a quantized-model, lowering our data storage needs and making it a more light-weight solution [9].

IV. Dataset

In this work, we use the ADE MIT Scene Parsing Benchmark Dataset [10]. The dataset comprises more than 22,210 images of indoor and outdoor scenes ranging from living rooms, bathrooms, suburban streets, urban districts, and more. Each image is provided with a respective segmentation image, comprising of colors mapped to 150 total semantic categories known as classes. These classes encompasses the most common objects encountered daily. Such as: tree, bed, mountain, wall, sky, and box. The segmentation image is the exact dimension of the image it is representing. The difference is instead of the original red, blue, and green (RGB) pixel values, the segmentation images hold unique colors that map to each class.

First, we familiarize ourselves with the dataset by writing auxiliary code to acquire, plot, and edit every instance. We observe that some segmentation images have unclassified labels. These labels, seen as pixels, are missing information are considered instances of the *invalid class*. Some instances have large chunks of invalid pixels, commonly on certain objects that cannot be categorized by the 150 classes, as seen in Fig. 1. Some instances have "hairline cracks" of invalid pixels where the segmentation between two different classes are not quite flush together. We deal with invalid pixels in two ways: (1) removing the instance completely from our dataset or (2) interpolating the data.

Because interpolating on large chunks of invalid pixels causes inaccurate segmentation maps, we remove instances with many invalid pixels by using quantile-based binning to extract the percentage of invalid pixels in each bin. We decide to cull the top 20% of instances with the most invalid pixels, giving us a threshold percentage of invalid pixels to cull at 12.58%. After culling, the dataset is reduced to 17,768 instances. Then we resolved the instances with fewer invalid pixels by applying nearest-neighbor interpolation onto the segmentation image, filling the smaller chunks and hairline cracks of invalid pixels with proper class values. Notice on Fig. 1, the hairline cracks and invalid pixel chunks should be small enough in size to interpolate accurately.



(a) (b) (c)

Fig. 1: An instance from the uncleaned ADE MIT Scene Parsing Benchmark Dataset. (a) original image with RGB channels (b) segmentation image with classes that map to its respective color (c) segmentation image with only the invalid classes masked in black

Next, we normalize the dimensions of all the original images and their respective segmentation images to 256x256 pixels. We chose this dimension as it was the standard input dimension for the neural network models. We applied a square-crop based on the minimum value between the width and height of the original image. Then we applied a bi-linear interpolation to shrink the images down to 256x256 pixels. The shape of the original image and segmentation image is the same: *(height, width, 3)*, where three represents the RGB color channels. Instead of requiring three channels to label a pixel, one 8-bit channel is enough to represent the 150 different classes. For every segmentation image, we created a new numpy array referred to as a segmentation map of shape: *(height, width, 1)*. One represents the direct class-value of the pixel. The values of the segmentation map will be the target values for our models.

The dataset has a scene label to categorize each image. Examples of scene categories are: *bathroom*, *countryroad*, *street*, and *library*. We decide to further reduce the dataset by choosing only images of indoor scenes. We exclude outdoor images because it becomes too much data to process with the hardware we had available to us, which is limited to an Intel(R) Core(TM) i7-4790K CPU and a GeForce(R) GTX 1070 GPU. Also, we deem it appropriate that separate models should be created: one for indoor navigation as done in this project

and one for outdoor navigation due to the high dissimilarities between the environments.

While the indoor instances can be isolated using binning on these scene categories mentioned previously, we find that much of the images have poorly judged scene categories designated to them. For example, a scene labeled *library* would showcase both the inside of a library room (indoor) and the entire library building (outdoor). Instead of discriminating between an indoor and an outdoor scene using scene labels, we determine an indoor scene by how much certain class pixels exist in the segmentation map. First, we use a histogram to determine the most common class objects. The three most common are *wall*, *building*, and *sky*. Then we apply Pearson correlation to find that many indoor-labeled scenes have high correlation with the class *wall*. Conversely, many outdoor-labeled scene categories have high correlation with the classes *building* and *sky*. We decide that instances with more *wall* pixels than *building* and *sky* pixels are indoor instances. From considering only indoor scenes with this process, our dataset is reduced to 10638 instances.

Finally, we transform the target feature from categorical to binary; we are only interested in two possible class labels for semantic segmentation: *non-obstacle* and *obstacle*. The 0 label represents the class *obstacle* and the 1 label represents the class *no obstacle*. We analyze the target feature correlations and observe that *floor* and *carpet* are the most highly correlated classes that are not obstacles. Thus, those two classes can be binned into the new *non-obstacle* class. The other classes are assumed to be obstacles and is subsequently binned into the new *obstacle* class. This remapping of values for every segmentation map simplified the dataset for our models.

V. Machine Learning Approach

A. Feature Imputation with Filters

Initially, the only descriptive features of the dataset include three RGB color channels of the original image. We impute more features using several image processing filters. First, we apply a grayscale feature. Then we apply a Canny Edge filter to emphasise edges between objects and the floor especially. We also apply blurring filters such as Sobel, Laplace Reflect, Gaussian-Blur, and Median -Blur to reduce the noise of detailed objects.

Of course, these filters do not account for the spatial aspect of pixels. Pixels that are in closer proximity to each other have a higher chance of belonging to the same material of an object. Therefore, we apply a Gabor filter to analyze the frequency of particular textures in localized regions. Finally, we expect images to be the point of view of a person walking with his or her eyes looking straight ahead. Consequently, the walk-able areas with no obstacles will most likely be the floor, which have a tendency to be located in the bottom rows of the image. We apply a column and row filter as my last imputations. After filtering, the number of descriptive features increased from three to 45.

B. Training

We split the dataset, consisting of 10638 instances using randomized, stratified sampling in preparation for training. The stratas are determined by the type of scene an image is categorized into. The train-validation-test split ration is 80-10-10. The number of training instances is 8273. The number of validation instances is 1182. The number of test instances is 1183. The traditional machine learning models we train are: (1) Logistic Regression, (2) Support Vector Machine (SVM), and (3) Random Forest.

There are limitations for training these models in terms of the amount of data and speed. Due to the increase of descriptive features due to the filters, our dataset size grew by a factor of 15. Also, each row of the dataset represents features of a single pixel. Since we scale our segmentation map size to 256x256, each semantic map would carry $256 \cdot 256 \cdot 45$ datapoints. Accounting for the 8273 instances, our dataset would have a total of $8273 \cdot 256 \cdot 256 \cdot 45$ datapoints. The final number of instances of our imputated dataset would be 542179328, each with 44 descriptive features and 1 target feature. The training time of these models is exponential to the amount of input data provided, so using all instances available is not feasibly due to time and memory constrains of our hardware.

C. Models

We chose 6 models for this project: 3 traditional and 3 Neural Network approaches. The Logistic Regression model was chosen as our simplest binary classifier. It is trained on 256 instances, which takes approximately 2 hours. SVM takes especially long to train as the amount of data provided increases. To reduce the training time, we use a linear kernel, training only on 32 instances. This takes approximately 2 hours to train. Only few training instances are needed to gain reasonable segmentation for the traditional machine learning models. Through grid-searching, we find that the best number of trees for our Random Forest model is $k = 5$ for 256 instances. This takes 3 hours to train. We are able to extract a list of the most important features for Random Forest. The top five important features based on information gain in order starting from the most were: *row*, *col*, *red*, *blue*, and $gaussian_7$. This shows that the models heavily depend on the spatial features of the image comparatively to the colors and textures. Note that the small subset of training instances are chosen to train each model within a reasonable amount of time. Perhaps to train these models with a larger dataset and faster training times, we could apply dimensionality reduction and consider only the top five most important features of our descriptive features instead of all 45.

VI. Neural Network Approach

A. EdgeTPU

Google has developed a application-specific integrated circuit (ASIC) that is specialized for neural network. In particular, the EdgeTPU is an ASIC designed to be operated on edge devices. Traditionally, many neural network computations for

applications goes through a process of sending input data into a cloud server. Then more robust computing systems would calculate the output and send it back to the client or device.

Alternatively, the EdgeTPU enables high-speed computations to be executed on the device itself, eliminating the need of sending the data to the cloud. The EdgeTPU prioritizes high volume computations and trades off with low precision. Consequently, the neural network models that are used by the EdgeTPU must be specially trained to limit the number of computationally-intensive floating point operations in place of simple integer operations. These integer precision may be as low as 8-bits. Compared to 64-bit floating point architectures such as the Intel Compute Stick, the EdgeTPU achieves lower power consumption, memory size, and latency [11]. Though, accuracy of these EdgeTPU-compatible models are often lower than their original counterpart. However, the crucial trade-off is lower inference times (latency) for the negligible accuracy loss in our semantic segmentation application.

B. DeepLabV3

DeepLabV3 is an architecture that uses artrous convolutions and Artrous Spatial Pyramid Pooling (ASPP) to preserve information of the original image throughout the layers of a backbone network [12]. First, it extract features from the backbone network. Then it uses atrous convolution, known as dilated convolutions, in the first few layers of the backbone to control the size of the feature map. The feature maps are fed into the the ASPP network to classify the pixels to their respective classes. Finally, the ASPP output is passed through a final convolution to resize the segmentation map to match the original image size.

C. Training

The neural network models are trained on the same dataset used for the traditional machine learning models mentioned in the previous section. However, there are no filters nor imputations required. The target features are simply the RGB channels and implicitly the position of the pixels. We also use the identical train-validation-test split.

We use the DeepLabV3 training pipeline library to create many checkpoints. The loss function for all models is binary cross-entropy. For validation data, the best model checkpoint is determined by the lowest *precision loss*. Since the width and height of our images is 256, the DeepLabV3 training parameter, *eval_crop_size* is set to 257. The reason is that the Tensorflow documentation calls for the *eval_crop_size* to be at least one pixel larger than the largest image in the training dataset. Other training parameters such as atrous rates and learning rates are suggestions given by the DeeplabV3 documentation. The batch size is determined simply by our hardware constraints, as we choose the highest batch size that can fit into memory. Note that these parameters may be further optimized. The backbone networks that we train are: (1) ResNet, (2) MobileNetV2, and (3) MobileNetEdgeTPU.

1) ResNet: ResNet stands for Residual Network. Its architecture is composed of residual blocks. Residual blocks are layers that directly connect to deeper layers whilst skipping the intermediary layers. This is known as *skip connections*. Stacking these residual blocks helps reduce vanishing gradients. Layers that are not conducive to the overall model may be skipped. The model was trained with an initial learning rate of $1 \cdot 10^{-5}$ and decays. The batch size is 4. We specify atrous rates of 6, 12, and 18. The number of steps for the best model based on the validation data is 60,000 before both the training and test loss curve plateau. This occurs when the learning rate is too high after a certain number of steps, so a lower initial learning rate or a sharper decreasing learning rate may yield better accuracy for this model.

2) MobilenetV2: The MobileNetV2 is based on inverted residual networks. The residual connections are connected between bottleneck layers instead of skipping them. The model is trained with an initial learning rate of $3 \cdot 10^{-5}$ and decays. The batch size is 8. The number of steps for the best model based on the validation data is 150,000 from early stopping. Fig. 2 shows that the validation curve has yet not diverged completely from the training curve. Therefore, continuous training beyond these steps may lead to a more accurate model.

3) MobileNetEdgeTPU: The MobileNetEdgeTPU is also based on inverted residual networks. The theoretically advantageous this model has over the MobileNetV2 is replacing in layers that better utilize TPU hardware. The model is trained with an initial learning rate of $3 \cdot 10^{-5}$ and decays. The batch size us 8. We specify an ASPP Convolution filter of size 256. The number of steps for the best model based on the validation data is 200,000 due to early stopping. Like the MobileNetV2 model, further training steps may yield lower losses. In fact, the accuracy metrics and latency between MobileNetEdgeTPU and MobileNetV2 is reasonably close, and optimizing their learning parameters may cause one model to outperform the other.

After training and achieving the best checkpoints for each model, we convert it to frozen graphs. The frozen graphs are then further converted to TensorFlowLite files. These are light-weight TensorFlow models. However, the floating-point datatypes used in the models cannot be efficiently used by the EdgeTPU. We need to convert the TensorFlowLite models to carry simple and small fixed-point computations. *Post-quantize-aware training* is an optimization that converts 32-bit floating-point numbers of the model's weights and activation outputs into the nearest 8-bit fixed-point numbers. This yields smaller models, faster inference speed, and less power consumption due to the lack of complex calculations. There are several techniques of post-quantize-aware training. The simplest technique, and the one we use, is *Full Integer Quantization*. We apply this by feeding the TensorFlowLite with a representative dataset to calibrate the minimum and maximum range of all floating-point tensors in the model. The representative dataset we use is 256 random instances from the validation dataset. We choose 256 as a middle ground between the suggested range of 100 and 500 instances

TABLE I: Evaluation Metrics on the Test Set

	TP	*TN*	*FP*	*FN*	*accuracy*	*precision*	*mIOU*	*AUC*	*latency*
Logistic Regression	14046659	543889	1958262	425783	85.9552%	56.0900%	18.5759%	0.593974	11.17444ms
Support Vector Machine	12629829	1650788	851363	1842613	84.1294%	47.2545%	37.9949%	0.766214	13.13960ms
Random Forest	14007531	610621	1891530	464911	86.1178%	56.7739%	20.5800%	0.605957	194.275ms
ResNetV1	27199152	180053616	16679790	13455623	87.3054%	61.9868%	47.4393%	0.792122	337.816ms
MobileNetV2	204427728	25073048	5062372	2825040	96.6774%	89.8737%	76.0700%	0.909191	57.8207ms
MobileNetEdgeTPU	204014224	22977736	7157670	3238608	95.6206%	87.6466%	68.8492%	0.873428	57.2326ms

from TensorFlow's documentation. The inputs and outputs of the TensorFlowLite are also quantized to 8-bit integers. After completing this process, the TensorFlowLite model is compatible with the EdgeTPU. Note that note all computations of the new model will be used by the EdgeTPU. Some may still require the CPU, which may cause bottleneck and higher latency.

Further, we co-compile each CPU-compatible semantic segmentation model with an object detection model into EdgeTPU-compatible models. The object detection model we are using is the SSD MobileNetV2, provided by Google. The best performing pair of segmentation and detection models in terms of recall and inference time is: MobileNetV2 Segmentation and SSD MobileNetV2. Then with EdgeTPU co-compiling, we redistribute 71 out of the 72 operations of the segmentation model from the CPU to the EdgeTPU hardware. We also redistribute 108 out of the 111 operations of the object detection model from the CPU to the EdgeTPU hardware.

VII. Results

Once a sensible overall model accuracy is achieved by the binary cross-entropy loss function, *precision* is the paramount metric for determining the best model. *Precision*, shown in Eq. 1, measures the purity of the positive predictions, *non-obstacles*, relative to the ground truth. The best performing model chosen is the one that maximizes the *precision* (or rather minimizes the *precision loss* shown in Eq. 2). In this project, we care more about the precision of the *obstacle* class. Erring to the side of the user's safety is warranted. A walkable area is said to be an area that has no obstacles. The case where an obstacle is erroneously predicted to be a walkable area, when it is not, is more critical than the case where a walkable area is erroneously predicted to be an obstacle. The trade-off is limiting the person's walkable area to ensure that he or she does not run into a potentially dangerous obstacle.

$$precision = \frac{tp}{tp + fp} \tag{1}$$

$$precision\ loss = 1 - precision \tag{2}$$

Another important metric is the pixel *accuracy*. *Accuracy* gives a simplistic assessment of our models: how many segmentation predictions are correct out of the all possible predictions in the image. However, this number may be disingenuous towards the usefulness of a model. For example, the logistic regression model is heavily under-fitted; its segmentation prediction naively consists of almost entirely *obstacle* predictions. Since the images of our dataset is composed of mostly head-height perspective, the walkable floor will only cover a small minority of the images at the bottom. The rest will be obstacles. Therefore, assuming everything is obstacles yields a deceptively high accuracy of 85.9552% for the logistic regression model as seen in Table I. That is due to the imbalance of data between the floor and the rest of the objects in each image.

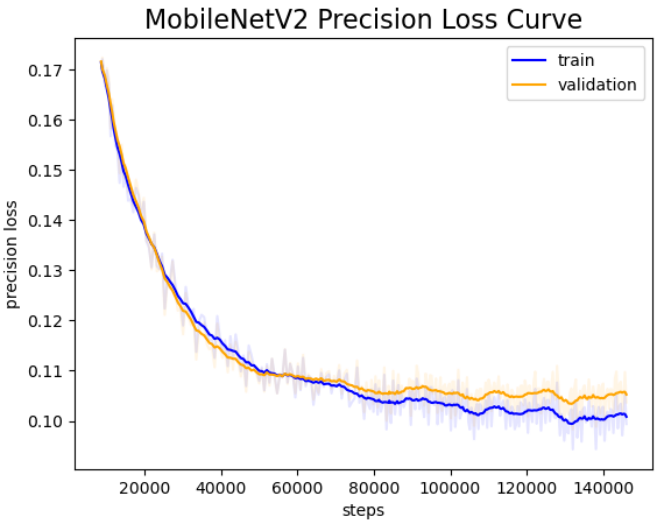


Fig. 2: The Precision Loss Curve measures the Precision Loss of our best performing model in terms of precision, MobileNetV2. The blue line shows the loss on the training data. The orange line shows the loss on the validation data.

The third accuracy metric we use is the Jaccard index, which is also known as the Intersection over Union measurement (IOU). This metric measures the percent overlap between the target mask and the prediction output. For each class, we calculate the Jaccard index and averages the results to achieve a mean IOU (mIOU).

The fourth and final accuracy metric is area under the receiver operating characteristic (ROC) curve (AUC). AUC gives a numerical measurement of how well a model can distinguish between the *obstacle* and *non-obstacle* classes. Along with the AUC score, we plot the ROC curves of the six models too. This visually shows the correlation between the true positive rate and the false positive rate as seen in Fig. 3.

The correlation matrix is an intuitive visual representation of a model's accuracy. For instance, we can see that all the

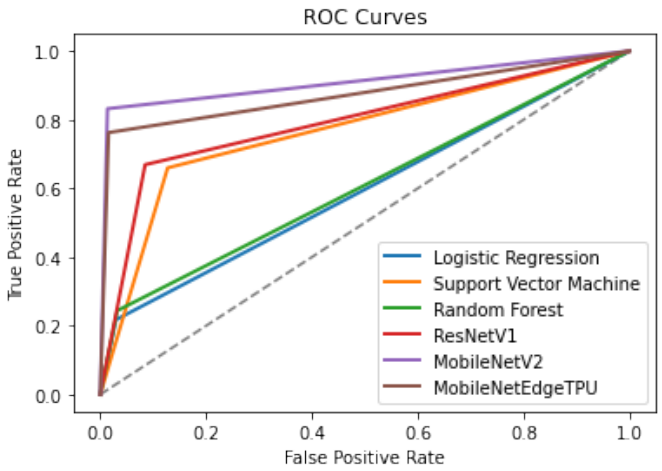


Fig. 3: The ROC curves of the six models. The x-axis represents the false positive rate. The y-axis represents the true positive rate. The middle dashed line represents a model that of purely random predictions. The ROC curve with the highest area under it belongs to MobileNetV2.

models predict *obstacle* well relative to *non-obstacle*. This is shown with a bright yellow color in Fig. 4. We can also see that the SVM model has relatively high false-positives, which leads to safety concerns for the user as previously explained. Also notice that the diagonal line from the top left to bottom right of the MobilenetV2's confusion matrix is the brightest compared to the other squares. Also, note that the cumulative sum of the number of true positives, true negatives, false positives, and false negatives differ between the traditional machine learning models and the neural network models. This is due to the smaller dataset used to train, validate, and test the traditional machine learning models. However, the colors in the correlation matrix provides a visual normalization of these values.

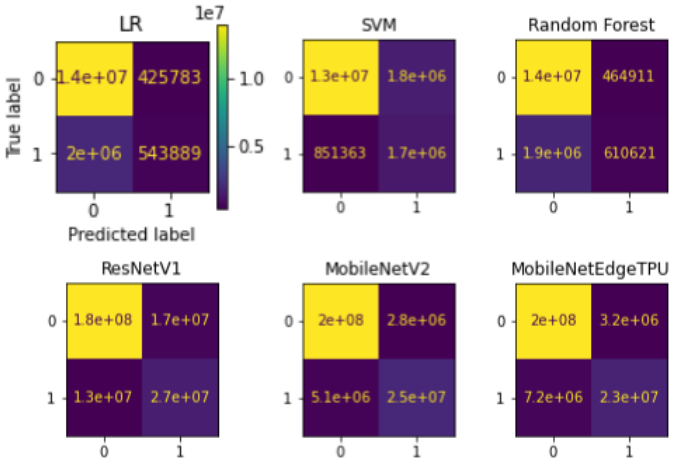


Fig. 4: The confusion matrices of the six models. The x-axis is the Predicted label. The y-axis is the True label.

The *latency* metric measures the speed of processing an image and formulating its respective semantic segmentation map. We co-compiled each CPU-compatible semantic segmentation model with an object detection model into EdgeTPU-compatible models. The object detection model used was the SSD MobileNetV2, provided by Google. The best performing pair of segmentation and detection models in terms of inference time and accuracy was: MobileNetV2 Segmentation and SSD MobileNetV2. Using this combination, we first tested the CPU models on the Intel® Core™ i7-11700K Processor and achieved inference times of 1200 milliseconds on average. Then with EdgeTPU co-compiling, we redistributed 71 out of the 72 operations of the segmentation model from the CPU to the EdgeTPU hardware. We also redistributed 108 out of the 111 operations of the object detection model from the CPU to the EdgeTPU hardware. We tested the EdgeTPU combination counterpart on the EdgeTPU and achieved inference times of 65 milliseconds on average–57 milliseconds for running the semantic segmentation model and 8 milliseconds for running the object detection model.

In summary, the best performing model is the MobileNetV2 model across all metrics we define *precision*, *accuracy*, *mIOU*, and *AUC*–excluding *latency*. However, the *latency* between the MobileNetV2 and MobileNetEdgeTPU model shown in Table I is negligible for our application. Examples for each model's semantic segmentation is show in Fig. 5.

Fig. 5: Examples of predicted semantic segmentation from the six models. Instances of the *non-obstacle* class are represented as yellow pixels. Instances of the *obstacle* class are represented as pink pixels.

REFERENCES

[1] J. Steinmetz et al., "Causes of blindness and vision impairment in 2020 and trends over 30 years, and prevalence of avoidable blindness in relation to VISION 2020: the Right to Sight: an analysis for the Global Burden of Disease Study." The Lancet. Global health vol. 9,2 (2021). [Online Serial]. Available: https://pubmed.ncbi.nlm.nih.gov/33275949/

[2] Austin's White Cane Day Celebration, "Canes" WhiteCaneDay. http://whitecaneday.org/canes/ (accessed Nov. 29, 2022)

[3] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, "Image Segmentation Using Deep Learning: A Survey," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 7, pp. 3523-3542, 1 July 2022, doi: 10.1109/TPAMI.2021.3059968.

[4] Z. -Q. Zhao, P. Zheng, S. -T. Xu and X. Wu, "Object Detection With Deep Learning: A Review," in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, doi: 10.1109/TNNLS.2018.2876865.

[5] K. Park, et al., "Real-time Mask Detection on Google Edge TPU" (2020), arXiv preprint arXiv:2010.04427 (2020).

[6] Abdullahi Madey, Ahmed Sheikh and Yahyaoui, Amani and Rasheed, Jawad,2021 International Conference on Forthcoming Networks and Sustainability in AIoT Era (FoNeS-AIoT), Object Detection in Video by Detecting Vehicles Using Machine Learning and Deep Learning Approaches,2021, pp. 62-65,doi: 10.1109/FoNeS-AIoT54873.2021.00023

[7] V. Jabade, U. Nahata, N. Jain, A. Pandey and T. Paratkar, "Obstacle Detection and Walkable Path Detection," 2022 IEEE Delhi Section Conference (DELCON), 2022, pp. 1-5, doi: 10.1109/DELCON54057.2022.9753182.

[8] A. Yazdanbakhsh, K. Seshadri, B. Akin, J. Laudon, & R. Narayanaswami "An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks" (2021), arXiv preprint arXiv:2102.10423.

[9] S. Vaidya, N. Shah, N. Shah and R. Shankarmani, "Real-Time Object Detection for Visually Challenged People," 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), 2020, pp. 311-316, doi: 10.1109/ICICCS48265.2020.9121085.

[10] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba, "Scene Parsing through ADE20K Dataset," Computer Vision and Pattern Recognition (CVPR), 2017.

[11] J. Sengupta, R. Kubendran, E. Neftci and A. Andreou, "High-Speed, Real-Time, Spike-Based Object Tracking and Path Prediction on Google Edge TPU," 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2020, pp. 134-135, doi: 10.1109/AICAS48895.2020.9073867.

[12] J. P. Rogelio, E. P. Dadios, R. R. Vicerra, and A. A. Bandala, "Object Detection and Segmentation using Deeplabv3 Deep Neural Network for a Portable X-ray Source Model," Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 26, no. 5, pp. 842–850, 2022.