CS230

Haptic Vest: Image Classification on the Edge

Philip M. Pfeffer
Department of Electrical Engineering
Stanford University
philpfef@stanford.edu
CS229, CS230

Raul G. Dagir
Department of Electrical Engineering
Stanford University
rgdagir@stanford.edu
CS230

Abstract

We propose adapting existing computer vision architectures, MobileNet V1 and V2 (1), to run three-class image classification with low inference latency while maintaining acceptable accuracy on consumer edge devices, with the intention to build a haptic garment. We aim for <1Mb `tflite` models and for <1s inference latency on the Raspberry Pi 4 (2). We achieve 86.6% accuracy with a 333 KB quantised (`uint8`) retrained MobileNetV1 model in 0.014s and 87.9% accuracy with a 690KB quantised (`uint8`) retrained MobileNetV2 model in 0.013s.

1 Introduction

Imagine being visually impaired but still able to feel the movement of surrounding crowds and cars on your body. This can be achieved by wearing a 'haptic vest' that uses a camera to capture images, a computer vision model to process them and coin-sized vibration motors that vibrate the shirt to communicate the output. This paper describes the design of the computer vision model for such a garment. The input to our algorithm is a (128, 128) greyscale image for MobileNetV1 and a (96, 96) greyscale image for MobileNetV2. We then use a retrained publically-available convolutional neural network (CNN) architecture called MobileNet to output a predicted classification of the image. The model classifies the image into the classes ['person', 'car', 'neither'].

This model is evaluated on three metrics, where accuracy is the optimising metric, and model size and inference latency are satisficing metrics. We set the constraint that the `tflite` model must be <1Mb because edge devices have small on-chip memory. We set the constraint that the inference latency on our test device, the Raspberry Pi 4, must be <1s because the haptic vest use-case requires rapid interpretation of images to relay the classification to the user in real-time.

We achieve 86.6% accuracy with a 333 KB quantised retrained MobileNetV1 model in 0.014s and 87.9% accuracy with a 690KB quantised retrained MobileNetV2 model in 0.013s. The quantisation procedure converts model parameters of type `float32` to type `uint8`.

This work is conducted alongside a CS229 paper by Philip Pfeffer (3), which uses a similar dataset for binary classification and creates an 'oracle' ResNet34 model without size or latency constraints, whose three-class classification accuracy, 90.5%, is used in this paper as an estimate for the upper bound of the accuracy we can expect from a smaller model.

2 Related Work

Several architectures have been proposed for accurately classifying images on edge devices. Google's MobileNet, versions 1, 2, and 3 (1; 4; 5), are one such architecture. Facebook's SqueezeNet (6)

CS229: Machine Learning, CS230: Deep Learning, Fall 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

	Train	Dev	Test
Neither	31,949	3,759	1,880
Person	21,651	2,548	1,274
Car	880	801	801

	Train	Dev	Test
Neither	32,041	3,887	1,962
Person	21,810	2,671	1,410
Car	12,107	2,670	1,411

Figure 1: Left: Pure COCO. Right: COCO with Stanford Cars.

is another such architecture. This network is larger than the MobileNet network, but is capable of image segmentation. This means the model is too large for our application, though with more lenient size constraints, it could improve accuracy. Work has also been done building custom architectures, such as binarised neural networks whose parameters take on only values $\{-1, +1\}$ (7). This model drastically simplifies the weights in exchange for a small reduction in accuracy on simpler image datasets, such as MNIST.

3 Dataset

Curating the dataset was key to achieving the model's performance. Two datasets were used during development: one with only images from COCO and another with images from both COCO (8) and the Stanford Cars dataset (9).

3.1 Regularisation & Preprocessing

The Haptic Vest application constrains which COCO images should be used and how. We only want the vest to react when the object is close enough to the user to be relevant. As a proxy for proximity, we introduce a hyperparameter, `area_threshold_percentage` and only include images whose largest labelled object occupies an area of at least `area_threshold_percentage`. For this dataset, we set `area_threshold_percentage` = 5%. We then resized images to $(128, 128)$, converted from RGB to greyscale and regularised the output between [0,1].

3.2 Pure COCO vs COCO and Stanford Cars

Initially, we used the 80-class COCO 2017 dataset (8) to collect images in the categories ['person', 'car', 'neither']. This portion of the dataset was collected for the CS229 sister paper(3). However, images of cars were underrepresented in the dataset as shown in Figure 1. Additionally, many car images were mislabelled. This dataset achieved 74.5% on the validation set using MobileNetV1 without data augmentation. After hand-cleaning all car image labels, the accuracy only improved to 76.5%.

We then developed a dataset that used COCO images for the classes 'person' and 'neither' and used Stanford Cars for the class 'car'. Stanford Cars contains larger images of cars, which is better suited to the application. This allowed us to train, develop and test on images closer to the ground truth distribution. This also helped the data imbalance issue. The mixed dataset was used to achieve the results reported in the abstract and introduction.

4 Method

Taking advantage of previous advances made in small, low-latency neural networks, we retrain the MobileNet_V1 and MobileNet_V2 architectures (1) for the haptic vest application.

4.1 Models

We do so by removing the top layer of MobileNet, adding our own input layers, and including pooling, dropout, and dense layers. The final layer has a softmax activation, which makes the output of the retrained model a three-class vector. The average pooling layer was included to down sample the feature maps of each image by summarizing the the average presence of a feature. The dropout layer has been included to regularize the model in hopes of achieving better accuracy on the validation set and at test time. The whole network is too large to show, so is included in the GitHub link. In Figure

```
Model: "mobilenet_v1_0.25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
tf_op_layer_RealDiv_1 (Tenso [(None, 128, 128, 3)]     0
_________________________________________________________________
tf_op_layer_Sub_1 (TensorFlo [(None, 128, 128, 3)]     0
_________________________________________________________________
mobilenet_0.25_128 (Function (None, 4, 4, 256)         218544
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 771
=================================================================
Total params: 219,315
```

```
Model: "mobilenet_v2_0.35"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 96, 96, 3)]       0
_________________________________________________________________
sequential_1 (Sequential)    (None, 96, 96, 3)         0
_________________________________________________________________
tf_op_layer_RealDiv_1 (Tenso [(None, 96, 96, 3)]       0
_________________________________________________________________
tf_op_layer_Sub_1 (TensorFlo [(None, 96, 96, 3)]       0
_________________________________________________________________
mobilenetv2_0.35_96 (Functio (None, 3, 3, 1280)        410208
_________________________________________________________________
global_average_pooling2d_2 ( (None, 1280)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1280)              0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 3843
=================================================================
Total params: 414,051
```

Figure 2: Left: MobileNetV1 embedded. Right: MobileNetV2 embedded.

2, we show the base models embedded in our input- and output-layer modifications, and include the number of parameters for each model.

We train the model in two stages: first, with the weights of the base MobileNet frozen, and, second, with all the weights unfrozen. The results of the transfer learning and fine tuning will be explored further below.

MobileNets are CNNs, which means that they learn parameters in convolutional kernels that are convolved across their inputs. This approach allows the network to identify features that may indicate 'person'-ness, 'car'-ness and 'neither'-ness. The use of CNNs instead of fully connected network means the model is robust to translations of objects in images, it maintains an explicit hierarchical representation of features, and contains fewer parameters. The last point is the most important in this use case as it allows high performance with fewer parameters to store on the edge device. MobileNets futher computational efficiency by using depthwise convolutions combined with pointwise convolutions (1), which allow the model to store intermediate results, and improve their inference performance.

4.2 Loss function and Optimiser

Since this project is a multi-class classifier, we use the categorical cross-entropy loss, $\mathcal{L} = \sum_{i=1}^{m} -y_i \log \hat{y_i}$. Following best practice in retraining MobileNets, in the first stage, we use the Adam optimiser (10) to optimise our loss function and in the second step, we use the RMSProp optimiser.

To avoid overfitting, we implement early stopping with a patience of three epochs. This means that if the development set loss does not decrease beyond the current minimum loss for three epochs, the model stops training.

4.3 Data Augmentation

We implement data augmentation to improve the ability of the networks to generalise beyond their training set. Specifically, at every iteration, images are randomly flipped horizontally and randomly rotated by a factor of 0.1. As discussed in the results section, the retrained MobileNets do not suffer from a high variance problem (a large difference between training error and development set error), so improvements from data augmentation are negligible.

4.4 Quantisation

When converting models from TensorFlow to `tflite`, we quantise the weights. The quantisation procedure converts model parameters of type `float32` to type `uint8`. This has negligible impact on accuracy and reduces the size of the model by over two times.

5 Results

We achieve 86.6% accuracy with a 333 KB quantised retrained MobileNetV1 model in 0.014s and 87.9% accuracy with a 690KB quantised retrained MobileNetV2 model in 0.013s on the Raspberry

Pi 4. In this section, we give details about where our model succeeds and fails and analyse the results. Recall that we aim to optimise accuracy under the constraints that the model size is <1MB and that inference on one image takes <1s. Our models can be found on GitHub.

5.1 Benchmark comparison

The ResNet34 (11) developed for CS229 (3) demonstrated that an 90.5% accuracy was a reasonable upper bound on accuracy. This model was trained without size or latency constraints, meaning that we expect this approximately models the best-case accuracy for the constrained models we develop here.

5.2 Hyperparameter Tuning

We explored optimiser learning rates in the range 0.001 to 0.0001 and found that 0.0005 produced the best results, given in the introduction. We explored early stopping patience values between 2 and 4, and found that the former often stopped too abruptly and the latter caused the model to loes accuracy on the development set because it overfitted the training set. Therefore, we use a patience value of 3.

5.3 Model Accuracies, Size and Latency

We present the accuracies of MobileNetV1 and V2 with and without data augmentation. Note, we hypothesise that test set accuracies are marginally higher than training set accuracies are because of the Dropout layers added to the model, allowing it to generalise well.

	Train Accuracy (%)	Dev Accuracy (%)	Test Accuracy (%)
MobileNetV1 (no aug)	85.4	86.5	86.6
MobileNetV1 (aug)	84.5	86.1	87.0
MobileNetV2 (no aug)	86.2	87.1	**87.9**
MobileNetV2 (aug)	86.0	86.5	87.0

We also present the final sizes of the unquantised and quantised models for MobileNetV1 and MobileNetV2. Since data augmentation does not change the number of parameters in a model and is not used at inference time, there is no difference in size between models that use data augmentation and their counterparts.

	Total Parameters	Unquantised Size (KB)	Quantised Size (KB)
MobileNetV1	219,315	866	**333**
MobileNetV2	414,051	1,627	**690**

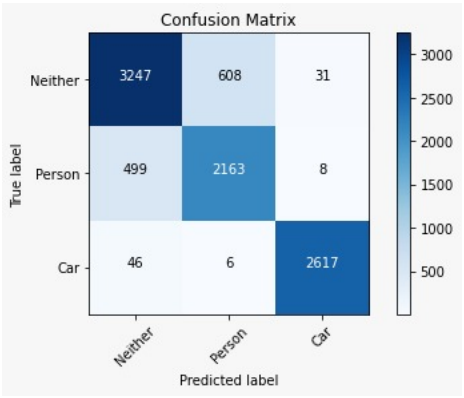
We also observe that data augmentation barely improves the performance of MobileNetV1. This can be explained by observing that the model variances are low, so data augmentation is unlikely to help as generalising is not a problem. On the other hand, our MobileNetV1 results indicate a high bias issue. We decided to increase the size of the model (and change architecture) to MobileNetV2. We observe that using the larger MobileNetV2 without data augmentation presents a 1.3% improvement over the smaller MobileNetV1 without data augmentation. Although within size constraint of the model, this minor improvement requires the number of parameters and quantised model size to be doubled form 333KB to 690KB.

	Time Taken (s)
MobileNetV1	0.009
MobileNetV1 (quant)	0.014
MobileNetV2	0.011
MobileNetV2 (quant)	0.013

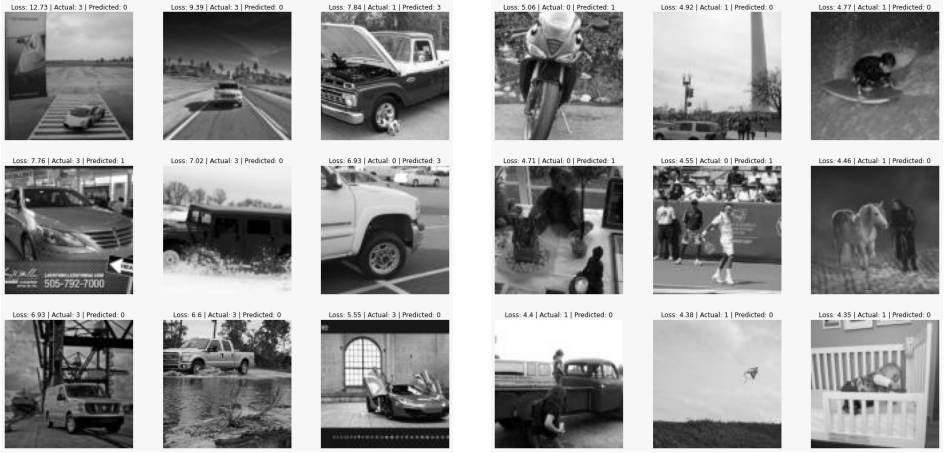
Finally, we present the average latency to process images in our development set, run using the Raspberry Pi 4. Using `tflite`'s Interpreter class for Python 3 we run a simple script on the Raspberry Pi to run all of our validation dataset on the model and ascertain the average classification time. We see that all models perform inference vastly faster than 1s, which satisfies our constraint.

5.4 Incorrect Predictions

We obtain the following confusion matrices for the highest accuracy model, MobileNetV2 without data augmentation. We observe that most of the incorrect predictions are due to the confusion of the 'neither' and 'person' classes. We hypothesise that this is because the images come from a similar distribution (COCO), whereas the car images are from another distribution. On further inspection, we also noted that the COCO images contain frequent mislabelling. For example an image of a hand, should not be considered a 'person', but COCO image segmentation does so.



We present the nine images with the highest loss predicted by our model (below left). Note that these are all images of cars, despite our high accuracy on cars. Where error is not due to mislabelling, as in the top right image, we suspect that the model is highly confident in its car predictions. Thus, when it is incorrect, it will be massively incorrect. To remove for this bias, we also present the nine images with the highest loss that are neither a car nor predicted to be a car (right). We observe that there is some mislabelling and that our model predicts objects such as teddy bears as 'person'. We suspect this is because it looks like a dark image of a person sitting down. Further observe that the image with the horse and person may have confused the model.



6 Conclusion

We achieve an 87.9% accuracy on the three-class classification of ['neither', 'person', 'car'], with a model size of 690KB and latency of 0.013s. This allows us to build a haptic garment capable of performing fast inference on the edge. With more time, we would want to clean the dataset by hand to improve labelling. We would also explore lower level representations of the model, such as running inference in C++ rather than in Python, which should further improve latency.

7 Contributions

We thank our TA advisor, Jo Chuang, for his advice. We acknowledge the work of Philip Pfeffer used for the CS229 paper (3), whose results helped guide our decisions and whose data we took advantage of.

References

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[2] "Raspberry pi 4 model b: 4gb ram." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/?resellerType=home

[3] P. Pfeffer, "Making haptic computer vision a bit shifty: Fast binary image classification on the edge," 2020.

[4] H. Chen and C. Su, "An enhanced hybrid mobilenet," pp. 308–312, 2018.

[5] J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. W. Leong, and P. Y. K. Cheung, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification," Cham, pp. 16–28, 2018.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016.

[7] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016. [Online]. Available: https://arxiv.org/abs/1602.02830

[8] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[9] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

Bibliography

Tsung-Yi Lin and Michael Maire and Serge Belongie and Lubomir Bourdev and Ross Girshick and James Hays and Pietro Perona and Deva Ramanan and C. Lawrence Zitnick and Piotr Dollár, "Microsoft COCO: Common Objects in Context". *arXiv cs.CV 1405.0312* (2015)

"`https://github.com/cocodataset/cocoapi/tree/master/PythonAPI`". *cocodataset/cocoapi* (Accessed 30 September 2020)

Hopkins Michael, Mikaitis Mantas, Lester Dave R. and Furber Steve, "Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations". *Phil. Trans. R. Soc. A.37820190052* (2020) `http://doi.org/10.1098/rsta.2019.0052`

"`https://www.coursera.org/learn/neural-networks-deep-learning/programming/XaIWT/logistic-regression-with-a-neural-network-mindset`", Coursera, *www.coursera.com*, (Accessed 30 September 2020)

Andrew G. Howard, Senior Software Engineer and Menglong Zhu, Software Engineer, "MobileNets: Open-Source Models for Efficient On-Device Vision". https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html, (2017)

Forrest N. Iandola and Song Han and Matthew W. Moskewicz and Khalid Ashraf and William J. Dally and Kurt Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size". *arXiv cs.CV 1602.07360* (2016)

Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition". *arXiv cs.CV 1512.03385* (2015)

"https://www.coursera.org/learn/neural-networks-deep-learning/ungradedLab/z0aIX/building-your-deep-neural-network-step-by-step", Coursera, *www.coursera.com*, (Accessed 30 September 2020)

Philip Pfeffer, "Making Haptic Computer Vision a Bit Shifty: Fast Multi-Class Image Classification on the Edge". *CS229 Milestone* (2020)

Jonathan Krause and Michael Stark and Jia Deng and Li Fei-Fei, "3D Object Representations for Fine-Grained Categorization", *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, (2013)

8 Appendix

8.1 Project Category

Computer Vision and Inference on the Edge.

8.2 Code

All of our code can be currently found at https://github.com/PhilipPfeffer/haptic_vest. This repository contains Jupyter notebooks for both the CS229 and CS230 projects. For this project, notable and corollary files include:

1. The ResNet 'oracle' notebooks written for the CS229 project, which we use as an accuracy benchmark for CS230.
2. The *data_pipeline.ipynb* and the *image_preprocessing.ipynb*, which are being used for data augmentation and image preprocessing.
3. The *mobilenet_retrain.ipynb* notebook, which contains the code for the model (with inspiration from *transfer_learning.ipynb*, in turn adapted from a notebook by Google, https://www.tensorflow.org/tutorials/images/transfer_learning). *mobilenet_retrain.ipynb* is the file we encourage you to read to understand our model setup, MobileNet_V1 retraining and frozen-unfrozen layers training process.