Semantic Image Segmentation for Mobile Platforms

Manasi Purohit
Electrical and Computer Engineering
mpurohit@andrew.cmu.edu

Samruddhi Pai
Electrical and Computer Engineering
sdpai@andrew.cmu.edu

Shaival Parikh
Electrical and Computer Engineering
shaivalp@andrew.cmu.edu

1 Problem Statement

Semantic Image Segmentation aims at mapping each pixel in an image to the corresponding class. With improvement in neural networks, the algorithms are becoming heavier to compute which creates a hardware barrier in case of embedded systems. In order to make these networks work on mobile platforms, we need to think of simple network architectures while maintaining the accuracy. The main objective of our work was to explore the existing network architectures that are computationally efficient such as MobileNetV2 and trained a DeepLabV3+ architecture over the base feature extractor for performing Semantic Image Segmentation. We also ported the trained model to a mobile device in order to test it's on device inference time.

2 Methods and Concepts

2.1 Model Architecture

MobileNetV2

We have used a pre-trained MobileNetV2 model as a base feature extractor. We trained a DeepLabV3+ model on the features extracted from MobileNetV2 to get class wise image segmentation output.
The architecture of MobileNetV2 is tabulated in Table: 1 and the block diagram for stride 1 and stride 2 bottleneck block is shown in Fig. 1. Here, t is the expansion factor applied to the input size, c is the number of output channels, n indicates how many times a particular operator (layer) is repeated and s is the stride. Output channels are the same for all the layers in a sequence. MobileNetV2 is computationally efficient and still gives better accuracy owing to the use of Depthwise Separable Convolutions, Inverted Residuals and Linear Bottlenecks.

Depthwise Separable Convolutions

Depthwise Separable Convolutions have a Depthwise Convolution layer followed by a 1×1 convolution layer also known as pointwise convolution. Depthwise Convolution layer applies a single convolution filter per input channel. Then the pointwise convolution builds new features by taking a linear combination of the input channels. This effectively reduces the number of parameters required as compared to standard convolutions.

Inverted Residual blocks

As opposed to regular residual blocks, inverted residual blocks connect the bottlenecks. Bottleneck layer, in general is a layer with fewer neurons as compared to it's previous and successive layers. Here, we have Depthwise Separable Convolutions as the bottleneck layer. These Bottleneck layers are

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Table 1: Model Architecture [5]

Input	Operator	t	c	n	s
224 x 224 x 3	conv2d	-	32	1	2
112 x 112 x 32	bottleneck	1	16	1	1
112 x 112 x 16	bottleneck	6	24	2	2
56 x 56 x 24	bottleneck	6	32	3	2
28 x 28 x 32	bottleneck	6	64	4	2
14 x 14 x 64	bottleneck	6	96	3	1
14 x 14 x 96	bottleneck	6	160	3	2
7 x 7 x 160	bottleneck	6	320	1	1
7 x 7 x 320	conv2d 1x1	-	1280	1	1
7 x 7 x 1280	avgpool 7x7	-	-	1	-
1 x 1 x 1280	conv2d 1x1	-	k	-	

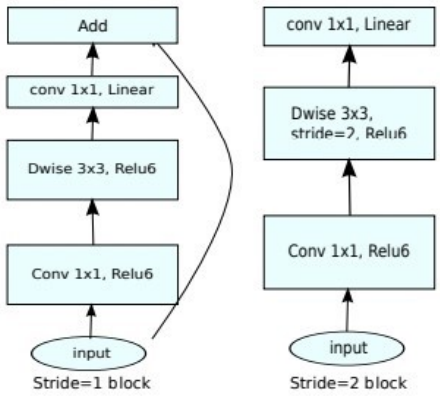

Figure 1: MobileNetV2 Architecture [5]

linear bottlenecks which mean that there is no activation or a linear activation applied on the layers shown in hatched lines in the Fig 3. Experiments have shown that removing non-linear activation such as ReLU has proven to improve the accuracy of MobileNetV2.

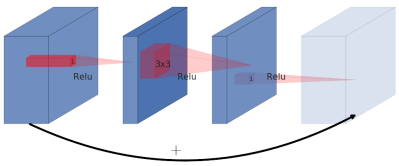

Figure 2: Residual Block [5

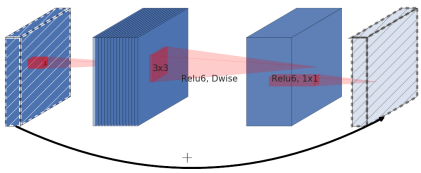

Figure 3: Inverted Residual Block [5]

DeepLabV3+

The DeepLabV3+ model architecture is as described in Fig.2. The DCNN in Fig. 4 uses MobileNetV2 in our proposed solution. DeepLabV3+ has an encoder-decoder based architecture. The encoder uses Atrous Convolutions also known as Dilated Convolutions which help to retain the feature map size as we go deeper into the network. It also uses Atrous Spatial Pyramid Pooling on image-level features which uses convolution kernels of different spatial dimensions in order to encode multi-scale contextual information. The decoder uses the low-level features from DCNN i.e MobileNetV2 and tries to improve the predictions at the boundaries of the objects. The decoder is an addition to the

previous DeepLabV3 architecture. We can control the Atrous (dilation) rate, which enables us to optimize the network parameters to make DeepLabV3+ hardware compatible.

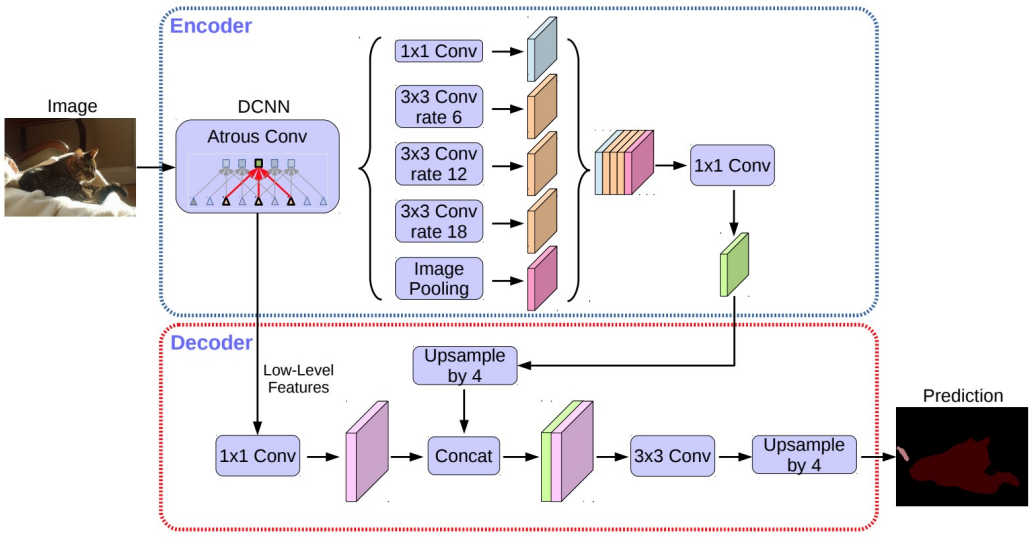

Figure 4: DeepLabV3+ Architecture [5]

Referring to Table: 1 and Fig. 1, we used a bottleneck block of input size 56^2 x 114 as our low-level feature extractor. We used second last layer as our high-level feature extractor whose size is 7^2 x 320 and is passed through various convolutional layers as shown in the encoder which is then upsampled to match the size of the low-level feature extractor.

2.2 Model Conversion

One of the objectives of this project was to deploy the trained model onto a mobile device and since the original Tensorflow model takes up more memory and is computationally heavy, it is not suitable for a mobile device. Hence, we convert the model into it's 'lite' version using the available Tensorflow APIs. This results in smaller storage size and less use of memory during execution and most importantly latency reduction. The model was optimized by converting the model parameters like the weights and the activations to a quantized data type. We experimented with a few different quantization schemes.

- **Post-training float-16 quantization:** This method involves quantization of the model weights of trained model from float-32 data type to float-16 quantized data type . This results in about 2x reduction in model size with minimal impact to accuracy.
- **Post-training int-8 quantization:** This method converts the activation outputs and weights of a trained model from float-32 data type number to the nearest int-8 number. This results in a smaller model with reduced inference time. This model supports int-8 data type inputs as well.
- **Post-training dynamic range quantization:** This method involves converting the weights and activation outputs from float-32 numbers to int-8 data type and float-16 data type numbers respectively. However, this strategy also checks for operations where activations are used and also support quantized kernels. If found, the activations are also dynamically converted to int-8 data type before processing and de-quantized post processing. This provides increase in speed without any change in accuracy of the model.

2.3 Android Application

After the 'lite' model generation, we deploy the model to an android application. The application was built using an open-source example code for android applications. The application captures images

from both the front and the back camera of an android device, creates a mask of the objects in the image and also classifies them into one of the 21 classes including the background that the model has been trained on. The application also calculates and displays the inference time and the time taken to generate a mask. The application was configured on a specific mobile device and based on it's computational capacity, we were able to run it on 4 parallel threads.

2.4 Data Preprocessing

We use Segmentation masks that have different categorical values for 21 different classes including background as one of the classes. Pascal VOC has Segmentation masks that have a color palette which needs to be mapped to categorical values. We have used the mapping as shown in Fig. 5. The categorical integer values for every class are the indices of these lists. The segmentation maps after mapping had values between 0 to 20 corresponding to each class including the background.

```
VOC_COLORMAP = [
    [0, 0, 0],
    [128, 0, 0],
    [0, 128, 0],
    [128, 128, 0],
    [0, 0, 128],
    [128, 0, 128],
    [0, 128, 128],
    [128, 128, 128],
    [64, 0, 0],
    [192, 0, 0],
    [64, 128, 0],
    [192, 128, 0],
    [64, 0, 128],
    [192, 0, 128],
    [64, 128, 128],
    [192, 128, 128],
    [0, 64, 0],
    [128, 64, 0],
    [0, 192, 0],
    [128, 192, 0],
    [0, 64, 128],
]
```

```
VOC_CLASSES = [
    "background",
    "aeroplane",
    "bicycle",
    "bird",
    "boat",
    "bottle",
    "bus",
    "car",
    "cat",
    "chair",
    "cow",
    "diningtable",
    "dog",
    "horse",
    "motorbike",
    "person",
    "potted plant",
    "sheep",
    "sofa",
    "train",
    "tv/monitor",
]
```

Figure 5: Pascal VOC Mapping

Figure 6: Original Image

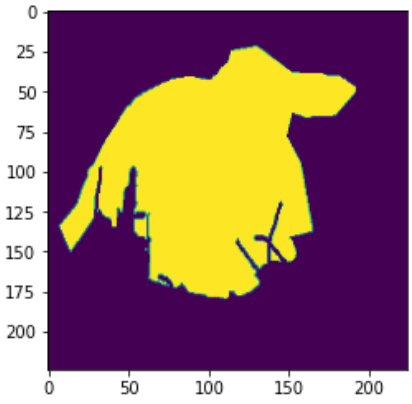

Figure 7: Mapped Segmentation Mask

An example of one of the original images from the Pascel VOC 2012 dataset is shown in Fig. 6 and its corresponding mask in shown in Fig. 7. We obsevered that the background covers significant mask area and hence decided converted the RGB mask to a categorial mask where background was

assigned as class 0, thereby creating a sparse mask. By using this method, we could generate sparse labels thereby aiding the computational efficiency.

3 Implementation and Results

We trained DeepLabV3+ model for performing class-wise image segmentation. The input to this model is the features extracted from a pretrained MobileNetV2. The segmentation labels present in Pascel VOC 2012 dataset maps each class to a unique RGB value as described in Sec. 2.4. As these modified labels were sparse in nature, we used sparse categorical cross entropy loss function, thereby leveraging the computational advantages of sparsity.

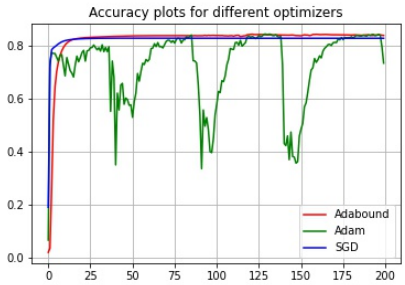

Figure 8: Accuracy plot for validation dataset

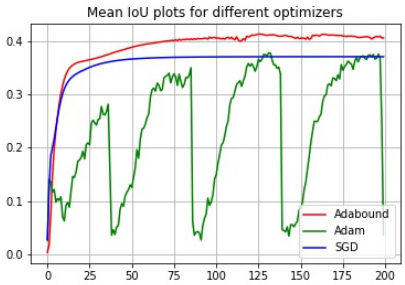

Figure 9: Mean IoU plot for validation dataset

Model training was performed using three optimizers - ADAM, SGD and AdaBound. As it can be seen from Figure: 8 and Figure: 9, with ADAM we got faster convergence but the model did not generalize well and the results were not stable. With SGD optimizer and Polynomial decay learning rate schedular, the mean IoU value increased to 40 on training dataset. The accuracy increased gradually and stabilized, giving comparable results on validation dataset. AdaBound, an optimizer that converges faster like ADAM and generalizes well like SGD, performed the best and could achieve a mean IoU of 43 on validation dataset. It was also observed that with increasing number of epochs, the model with ADAM and AdaBound optimizers overfitted faster than SGD.

For deploying the model on edge devices, we converted the float-32 Keras model to float-16, int-8 and dynamic range TFLite models using Tensorflow APIs. From the Table: 2 we can see that the accuracy and mean IoU value did not drop for TFLite models but we gained speed by a factor of 10. It can also be seen that the pixelwise model accuracy is much higher than the mean IoU values. This is because our model could predict the class labels correctly (including the background) but the outline of the generated mask were partially correct thereby having less intersecting area with the original mask.

Table 2: Model Metrics

Model	Accuracy	Mean IoU	Inference Time	Size
Original Model	82.98%	28.73	0.333 sec	36 MB
float-16 Model	82.90%	28.63	0.039 sec	15 MB
int-8 Model	74.33%	27.89	0.034 sec	3 MB
Dynamic range Model	83.17%	28.74	0.031 sec	6 MB

Fig. 10 is an example of on eof the original images from Pascel VOC 2012 dataset and Fig. 11 is its corresponding categorical mask. Fig. 12, Fig. 13 and Fig. 14 are the three masks generated by the keras and TFLite models trained by using SGD as optimizer and polynomial decay LR scheduler. These masks also justify the claim that converting to Lite models does not degrade the prediction capabilities. Background and sheep were correctly classified but the model also misclassified some part of it to be dog. This was maybe because darker/ low light areas in the input image can mislead the model.

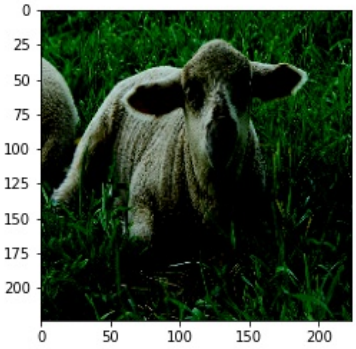

Figure 10: Original image

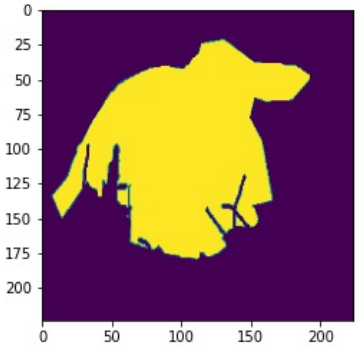

Figure 11: Categorical label mask

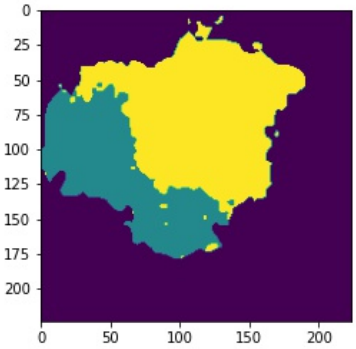

Figure 12: Mask generated by Keras model

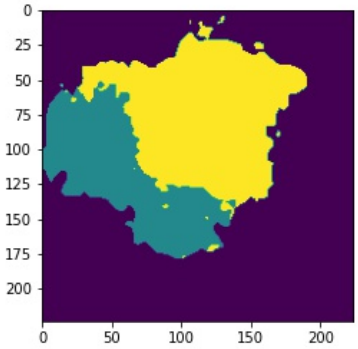

Figure 13: Mask generated by TFLite Float-16 model

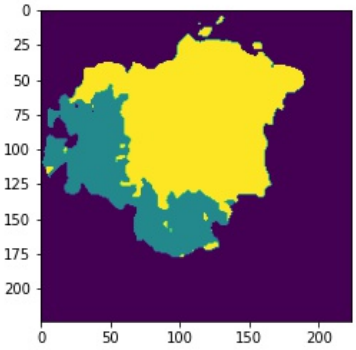

Figure 14: Mask generated by dynamic range model

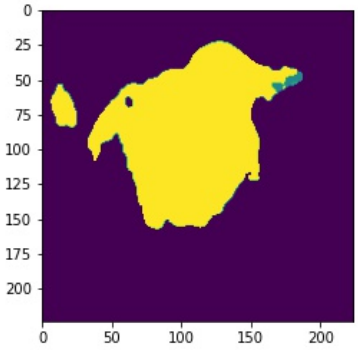

Figure 15: Mask for back camera image

After observing that the model's accuracy started to stagnate, we tried training the model with AdaBound as are optimizer to leverage the effect of both ADAM and SGD. Fig. 15 shows the mask generated by this model and it can be seen that training with AdaBound classification accuracy as well as mask generation was better.

Figures below show segmentation and object classification results from our android application. We observed that the model could segment and classify object from images taken in real time by front or back camera. The mask created was not perfect since we could not train the model enough, due to

lack of computational resources. Furthermore, the model segment well (seat of the chair in Fig. 20 could not be captured) in low lighting conditions as seen in Fig. 20 and Fig. 21.

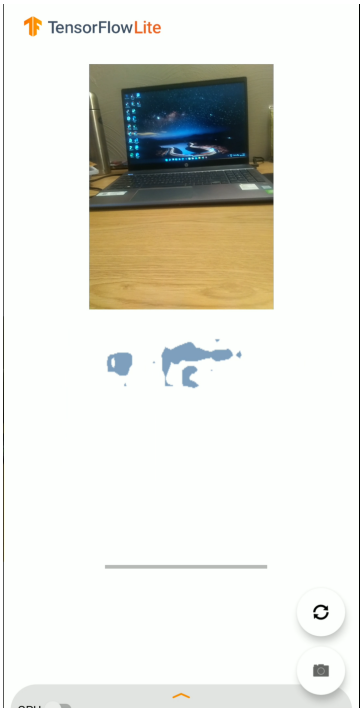

Figure 16: Mask for back camera image

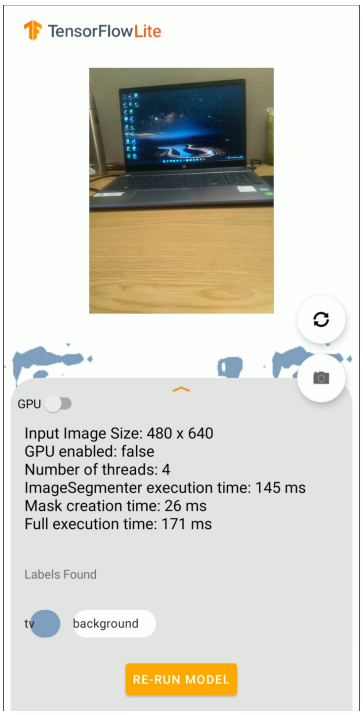

Figure 17: Classification and processing details for back camera image

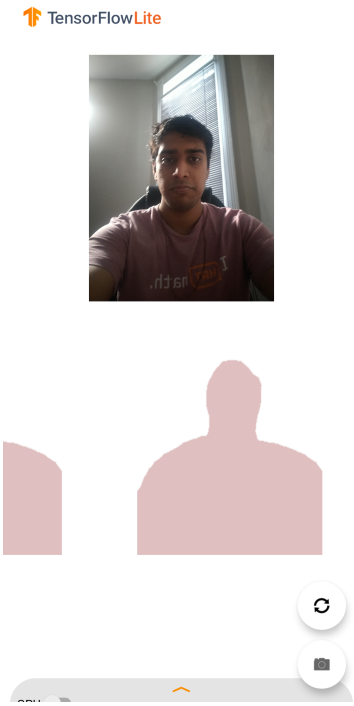

Figure 18: Mask for front camera image

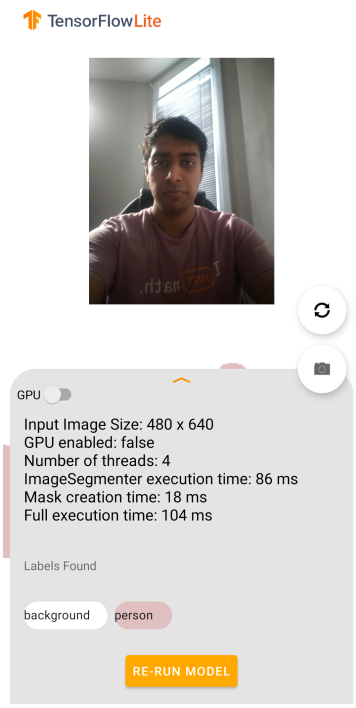

Figure 19: Classification and processing details for front camera image

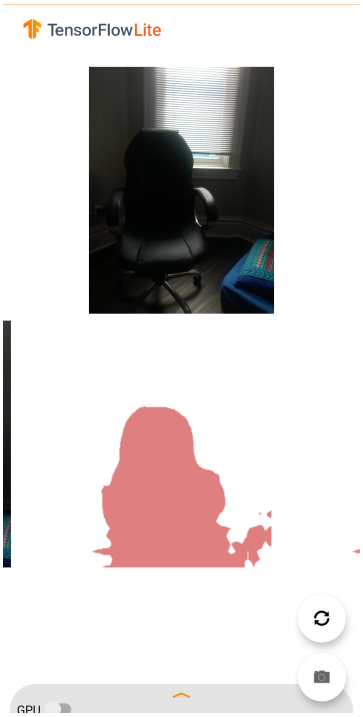

Figure 20: Mask for back camera image - low lighting

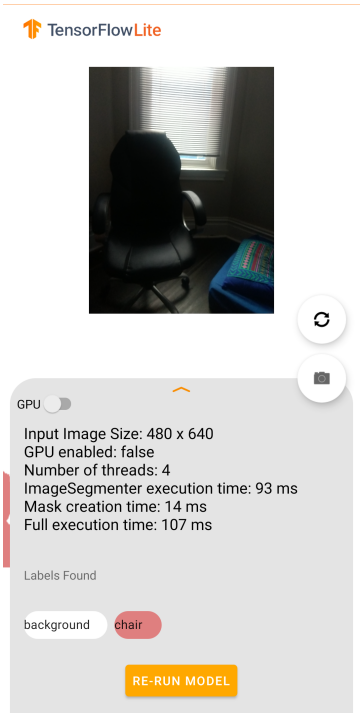

Figure 21: Classification and processing details for back camera image - low lighting

4 Conclusion

We trained DeepLabV3+ model for segmenting an image where we analysed the effect of different optimizers during the training step. We converted the original Keras model to different TFLite models and compared their performance. The dynamic range quantized TFLite model was deployed on an android device. The model could generate mask and predict the classes seen in the image taken by the front as well as rare camera of the device. Due to limitation of training resources, we could not train the model extensively for achieving better results.

5 Future Work

In our work we have used post-training quantization schemes. There are also ways to figure out quantization as we train knows as quantization-aware training. Going forward, we plan to implement and experiment qunatization-aware training in our work.

References

[1] Xiangyu Zhang,Xinyu Zhou, Mengxiao Lin, and Jian Sun. *Shufflenet: An extremely efficient convolutional neural network for mobile devices.* CoRR, abs/1707.01083, 2017.

[2] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. *The power of sparsity in convolutional neural networks.* CoRR, abs/1702.06257, 2017.

[3] Min Wang, Baoyuan Liu, and Hassan Foroosh. *Design of efficient convolutional layers using single intra-channel convolution, topological subdivisioning and spatial "bottleneck" structure.* CoRR, abs/1608.04337, 2016.

[4] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserma. *The Pascal Visual Object Classes challenge a retrospective.* IJCV, 2014.

[5] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam. *Rethinking Atrous Convolution for Semantic Image Segmentation.* https://arxiv.org/abs/1706.05587

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks* https://arxiv.org/abs/1801.04381

[7] Pascal Visual Object Classes dataset: `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html#data`

[8] MobileNetv2 source code. Available from: `https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet`.

[9] Post-training quantization of Tensorflow models: `https://www.tensorflow.org/lite/performance/post_training_quantization`