Semantic Segmentation Optimization Techniques for deployment on edge devices

Team: ms5945 (Madhavan Seshadri) and sk4653 (Shravan Karthik)

Abstract

Increased efficiency, power and memory are critical requirements to applications deployed on the edge device. Deep Convolutional Neural Networks contain convolutional and dense layers that have millions of trainable parameters in them. The disk utilization and memory footprint of such a model is extremely high which makes it infeasible to store and run on edge computing devices which have such constraints. Semantic Segmentation is the process of making pixel-wise classification of an image to categories associated with them. There are several standard models which are used for the implementation of this problem such as U-Net, Semi supervised learning approaches etc.

Semantic Segmentation has several applications in the health industry, where this can be used for tumor detection etc. The ability to deploy these models on edge devices with acceptable level of accuracy will be a tremendous leap in the portable health devices industry. In this project, we aim to take a step in this direction by developing a framework for facilitating model development and applying post training optimization techniques. We mainly focus on implementation of Fully Convolutional Network and U-Net, with the training dataset being MS-COCO and post optimization techniques of Sparsification and Quantization for improving model disk utilization and memory footprint. Our results show that the model obtained after sparsification and quantization reduces the disk utilization by 70%.

Introduction

Semantic segmentaion is the process by which every pixel from the image is assigned an object category coressponding to the class present in the image. Convolutional neural networks are interesting because they provide hierarchies of features in each of the layers. The idea is to take an image of n X n and predict a pixel wise classification for each of the pixels.

A starter level approach to predicting the objects in an image is using a bounding box detector. This can be considered as a superset to the semantic segmentaion problem. Several approaches for bounding box prediction model have been proposed in the past, such as (Simonyan et al.), (Szegedy et al.). A typical semantic segmentation problem is a combination of multiple-object detection with a bounding box occupying the smallest convex hull of the predicted object. Local correspondence problem is also used for predicting various objects in the image. Usually, SIFT features are detected from an image and used to localize with known SIFT features of other objects in the dataset. Convolutional Neural networks are also being used to solve this problem and such approaches are explored in (Fischer et al.) and (Long et al.).

Internet of Things (IoT) is gaining in popularity over the years with many new devices gaining intelligent learning techniques available to us. Several techniques are explored in (Li et al.). Interesting applications such as Localization of Epileptogenicity are explored in (Hosseini et al.) where dense deep learning models are used on edge computing devices to predict the Epileptogenicity of an MRI image. Increased number of approaches are possible with appropriate optimization done on trained models.

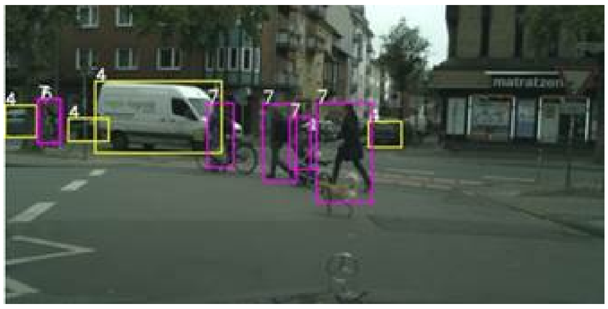
Optimization of deep learning problems is also gaining interest and several hardware level optimization solutions are also being researched. For example, (Bojnordi et al.) proposes a Boltzmann machine as a hardware accelartors for deep learning models deployed for achieving speedup. Specialized hardware such as an accelarator on FPGAs was developed in (Wang et al.). The DLAU accelerator employs three pipelined processing units to improve the throughput and utilizes tile techniques to explore locality for deep learning applications.

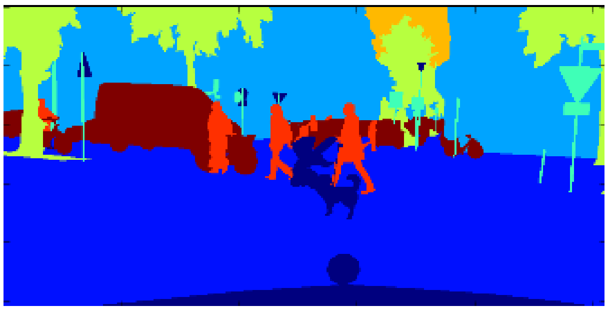
Several areas of applications exist for our Semantic Segmentation problem. Most notably, in autonomous vehicles Fig 1.. Semantic segmentation for autonomous vehciles using Point Cloud data was explored in (Wang et al.). (Treml et al.) proposes a novel architecture consisting of ELU activations, Squeeze-Net like encoder, and multiple parallel dilations specialized for embedded devices onboard a car. The claim is that the loss in performance is insignificant to the gain in speedup for this problem. Several datasets such as the Synthia dataset (Ros et al.) is also available to solve such a problem. The dataset consists of synthetic images for semantic segmentation of an urban scene.

With a strong motivation to deploy models on the edge, we decided to come up with a framework that can be used for building, training, optimizing and running comparison analysis on different models. We use optimization techniques such as Sparsification during training and Post Training quantization to reduce the model size and potentially speed up computation. In the following sub-sections, we will discuss the related work in the domain, followed by system architecture & setup required for our framework, models (loss) and optimization techniques applied and discuss our results.

Typical Use Cases

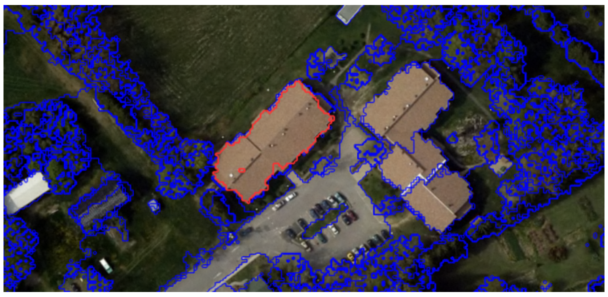
1. Autonomous Vehicles\





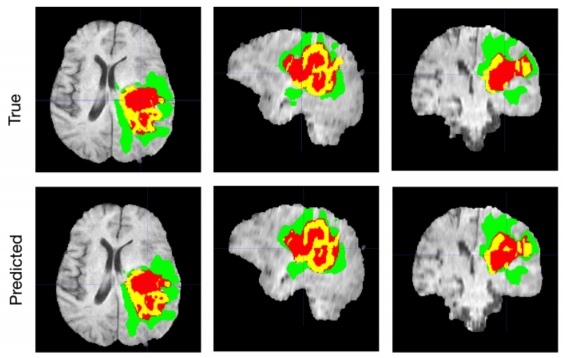
(Figure 1: https://blogs.nvidia.com/blog/2016/01/05/eyes-on-the-road-how-autonomous-cars-understand-what-theyre-seeing/ (https://blogs.nvidia.com/blog/2016/01/05/eyes-on-the-road-how-autonomous-cars-understand-what-theyre-seeing/))

2. GeoSensing: Satellite based Land Usage Efficiency



(Figure 2: https://gisgeography.com/wp-content/uploads/2017/06/OBIA-object-based-image-analysis-geobia-678x322.png (https://gisgeography.com/wp-content/uploads/2017/06/OBIA-object-based-image-analysis-geobia-678x322.png))

3. Medical diagnostic enhancements\



\ (Figure 3: https://news.developer.nvidia.com/automatically-segmenting-brain-tumors-with-ai/ (https://news.developer.nvidia.com/automatically-segmenting-brain-tumors-with-ai/)) and many more....

Literature Review

Our motivation is to apply Semantic Segmentation on edge devices and shall cover realted work in this domain.

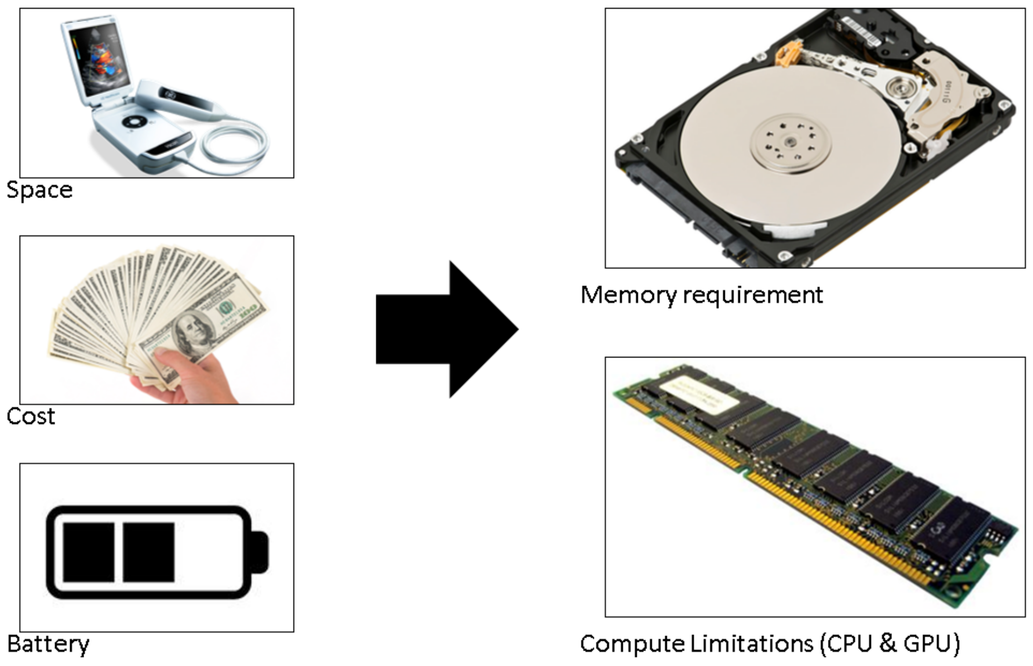
Using deep learning techniques for semantic segmentation has gained popularity in the recent years. Their insight is to build "fully convolutional" networks that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning (Long et al.). To achieve pixel wise classification, the authors propose replacing the last decode layer, with a transposed convolutional layer which helps upsample the image to provide a pixel wise mapping for each image. However, because the encoder (layers before transposed convolutions), decreases the input size by a factor of 32, the features aren't localized very well, and fine-grained segmentations aren't achieved easily. To overcome this limitation, (Drozdzal et.al) suggested using residual blocks instead of stacked convolutional layers, and using short skip connections. (Ronneberger et.al) suggest adding long skip connections between corresponding encoder and decoder layers. This introduces a symmetry during up-sampling and helps achieve precise localization. Expanding on this idea Jengou et.al CITE propose using dense blocks while maintaining the u-net architecture providing multi-scale supervision. This model is claimed to perform well in spatially dense prediction tasks.

(Girshick et al.) proposed RCNN which uses a selective region search to detect 2000 selective features which are called region proposals. Instead of searching through the entire space, the 2000 feature search space is warped into a vector and fed into a dense 4096 layer as an output. As an advancement to this apprach, (Kaiming et al.) proposes Mask RCNN for semantic segmentation generates segmentation masks for each instance of object in the image. The proposed regions from RCNN are then used to predict the class of the object and provide a bounding box for the detected object in the image. The model generalizes and can even be used for human pose detection. This model is extremely popular for segmentation tasks implemented on embedded devices there is computational advantage obtaining the 2000 feature space. Currently, Mask-RCNN has the best performance and is considered state of the art for semantic segmentation tasks.

Applications of semantic segmentation on edge devices is also gaining popularity. There are several approaches which have specified deep learning models for low power devices. (Mehta et al.) proposed ESPNet, for semantic segmentation of high resolution images under resource constraints. ESPNet is based on a new convolutional module, efficient spatial pyramid (ESP), which is efficient in terms of computation, memory, and power. The model is claimed to be 22 times smaller than PSPNet with a loss of 8% accuracy. Bahl et al. proposes a network C-UNet which is clamed to be 100k times smaller than the standard U-Net models.

As such, in a typical development scenario, we would need a framework which can compare and contrast multiple models and decide the best model for our production application

Compute and Memory Optimization Requirements



It is important to run these CNN models on on edge devices which have high constraints on compute, space and power. Thus, to run these models on edge devices, the compute and space occupied by these models must be reduced. For example, the trained model of Mask RCNN currently occupies 250MB on disk. In comparison, a FCN model occupies 90MB while a U-Net model occupies 10MB. Another measure of the network performance is multiply accumulations (MAC). For FCN, we need 238.6 Mega MAC, while for Mask RCNN we need 850 Mega MAC on a 128x128 3 channel image. For a U-net model we use 48.6Mega MACs (Alom et al.)

Dataset and pre-processing

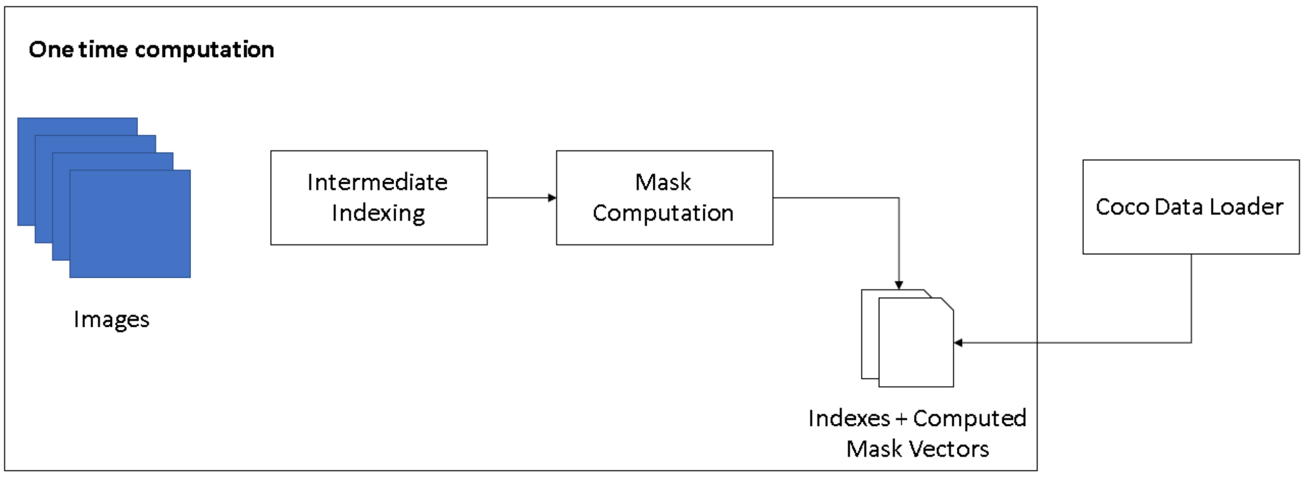
Training Dataset Description

The main training dataset for this problem was Microsoft COCO.

1. Dataset:
    - COCO is a object detection, segmentation and captioning dataset.
    - The segmentation dataset consists of ~120k training images and about ~5k validation images.
    - The test set of the standard split is divided into dev-test and challenge test.
    - The dataset consists of 91 classes of objects contained in them with a pixelwise prediction of the classes on the images.
    - The 91 classes of objects can be categorized into super-classes which are 13 in number.

For the purposes of this project, training was performed on `train2017` of MSCOCO dataset and the validation was performed on `val2017` of the corressponding dataset over 13 classes. The super-classes used are: 'background', 'appliance', 'electronic', 'accessory', 'kitchen', 'sports', 'vehicle', 'furniture', 'food', 'outdoor', 'indoor', 'animal', 'person'

Data Processing Pipeline



We run a one time encoding job and compute the masks for all the images in our dataset. We then load a Keras custom data loader to load images into memory batch by batch.

This is necessary as we would need to reduce our memory footprint of our training model.

class CocoDataGenerator(Sequence):

```
    def __init__(self, cfg, run_type):
        fetch_prefix = utils.prefixer.fetch_prefix(run_type)

        self.batch_size = cfg[fetch_prefix]["batch_size"]
        self.index_file = cfg["data"][fetch_prefix]["index_file"]
        self.dataset_dir = cfg["data"][fetch_prefix]["dataset_dir"]
        self.labels_dir = cfg["data"][fetch_prefix]["labels_dir"]
        self.image_width = cfg["data"]["dimensions"]["img_width"]
        self.image_height = cfg["data"]["dimensions"]["img_height"]
        self.image_num_chans = cfg["data"]["dimensions"]["img_num_chans"]
        self.num_images = file_line_count(self.index_file)

    def __len__(self):
        return int(np.ceil(self.num_images/float(self.batch_size)))

    def __getitem__(self, idx):

        batch_images = list()
        batch_labels = list()

        with open(self.index_file) as f:
            result = itertools.islice(f, idx*self.batch_size, (idx+1)*self.batch_
    size)

            for line in result:
                batch_images.append(cv2.imread(os.path.join(self.dataset_dir, lin
    e.strip())))
                label_mat= np.load(os.path.join(self.labels_dir, line.strip()+".n
    py"))
                label_mat = label_mat.reshape(label_mat.shape[0],label_mat.shape
    [1],1)
                batch_labels.append(label_mat)

        return np.array(batch_images), np.array(batch_labels)
```

To train our model, we perform the following changes:

Network Architecture

For semantic segmentation I implemented the Fully Convolutional Layer (FCN) architecture which comprises of the below layers:

```
x = Conv2D(
64,
(7, 7),
strides=(2, 2),
padding='same',
name='conv1',
kernel_regularizer=l2(weight_decay)
)(img_input)

x = BatchNormalization(axis=bn_axis, name='bn_conv1')(x)
x = Activation('relu')(x)
x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = self.convolution_block(3, [64, 64, 256], stage=2, block='a', strides=(1,
 1))(x)
x = self.identity_block(3, [64, 64, 256], stage=2, block='b')(x)
x = self.identity_block(3, [64, 64, 256], stage=2, block='c')(x)

x = self.convolution_block(3, [128, 128, 512], stage=3, block='a')(x)
x = self.identity_block(3, [128, 128, 512], stage=3, block='b')(x)
x = self.identity_block(3, [128, 128, 512], stage=3, block='c')(x)
x = self.identity_block(3, [128, 128, 512], stage=3, block='d')(x)

x = self.convolution_block(3, [256, 256, 1024], stage=4, block='a')(x)
x = self.identity_block(3, [256, 256, 1024], stage=4, block='b')(x)
x = self.identity_block(3, [256, 256, 1024], stage=4, block='c')(x)
x = self.identity_block(3, [256, 256, 1024], stage=4, block='d')(x)
x = self.identity_block(3, [256, 256, 1024], stage=4, block='e')(x)
x = self.identity_block(3, [256, 256, 1024], stage=4, block='f')(x)

x = self.convolution_block(3, [512, 512, 2048], stage=5, block='a')(x)
x = self.identity_block(3, [512, 512, 2048], stage=5, block='b')(x)
x = self.identity_block(3, [512, 512, 2048], stage=5, block='c')(x)

x = Conv2D(
    classes,
    (1, 1),
    kernel_initializer='he_normal',
    activation='linear',
    padding='valid',
    strides=(1, 1),
    kernel_regularizer=l2(weight_decay)
)(x)

x = BilinearUpSampling2D(size=(32, 32))(x)
```

In this architecture, there are four convolution blocks and 12 identity blocks. The proposed DAG learns to combine coarse higher layer information with fine lower layer information. In the Bilenear Upsampling step, stride 32 convolutions are reduced to pixel wise prediction in a single step. The implemented model has a 5 stage process and the model has a total of `23,534,915` trainable parameters.

A convolutional block is defined as follows:

```
filters1, filters2, filters3 = filters

if image_data_format() == 'channels_last':
    bn_axis = 3
else:
    bn_axis = 1

x = Conv2D(
    filters1,
    (1, 1),
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(input_tensor)

x = BatchNormalization(axis=bn_axis, momentum=momentum)(x)
x = Activation('relu')(x)

x = Conv2D(
    filters2,
    kernel_size,
    strides=strides,
    padding='same',
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(x)

x = BatchNormalization(axis=bn_axis, momentum=momentum)(x)
x = Activation('relu')(x)

x = Conv2D(
    filters3,
    (1, 1),
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(x)

x = BatchNormalization(axis=bn_axis, momentum=momentum)(x)

shortcut = Conv2D(
    filters3,
    (1, 1),
    strides=strides,
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(input_tensor)

shortcut = BatchNormalization(
    axis=bn_axis,
        momentum=momentum
    )(shortcut)
```

In a convolutional block, convolutions of multiple filter sizes are performed and a skip connection is added in the final step where the convolutional blocks in the middle are intuitively expected to learn the differences.

The identity block is defined as follows:

```
filters1, filters2, filters3 = filters
if image_data_format() == 'channels_last':
    bn_axis = 3
else:
    bn_axis = 1

convolution_base_name = 'res' + str(stage) + block + '_branch'
bn_base_name = 'bn' + str(stage) + block + '_branch'

x = Conv2D(
    filters1,
    (1, 1),
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(input_tensor)

if self.is_pruning_enabled:
    x = sparsity.prune_low_magnitude(x, **self.prune_params)

x = BatchNormalization(axis=bn_axis, momentum=momentum)(x)
x = Activation('relu')(x)

x = Conv2D(
    filters2,
    kernel_size,
    padding='same',
    use_bias=False,
    kernel_initializer='he_normal',
    kernel_regularizer=l2(weight_decay)
)(x)
```

Loss Function:

Intersection Over Union (IoU)

Intersection over union is a class-wise weighted value of intersection of the common area over the smallest convex hull of the geometry contating the predicted output vs the actual image. This was originally defined in (Rezatofighi et al.)

Intersection over Union (IoU) = Area of Overlap/Area of Smallest Convex Hull Union

1. Disadvantages
   - Since we are dealing with a lot of classes, computationally computing the bounding box, convex hull and calculating the loss is expensive
   - The function IoU is not differentiable and a loss cannot be propogated to the previous layers to minimize.
   - Note: There are some techniques to make IoU differentiable. Generalized IoU, released by Stanford in [1] certainly helps to consider this as a suitable loss function.

Sparse Categorical Crossentropy

Pixelwise prediction of the classes need to be compared with the true generated masks. The masks contain the super-classes from 0 to 13 (14 in number). It is used to measure the dissimilarity between the predicted classes and generate an associated loss for the same. The dimension of the last layer in the network must be the pixel wise prediction of the classes converted from the softmax layer which is typically used.

We need to minimize this associated loss function while training our network.

Categorical Crossentropy is defined as below:

$$L(y,\hat{y}) =- \sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log(\hat{y}_{ij}))$$

Intuitively, we are trying to maximize the probability of correct class prediction by minimizing the log of the probability prediction when the system outputs a correct class.

The advantage of the sparse categorical cross-entropy when compared to softmax cross-entropy, is for every category, there isn't a need for a one-hot vector. This helps reduce the size of the problem drastically, as for each image, the output can be represented as a matrix of dimension `width x height` , rather than representing the output as a tensor of `width x height x num_super_class` .

Optimizations

To optimize the model, two methods are employed:

- Sparsification during training
- Quantization post training

Sparsification

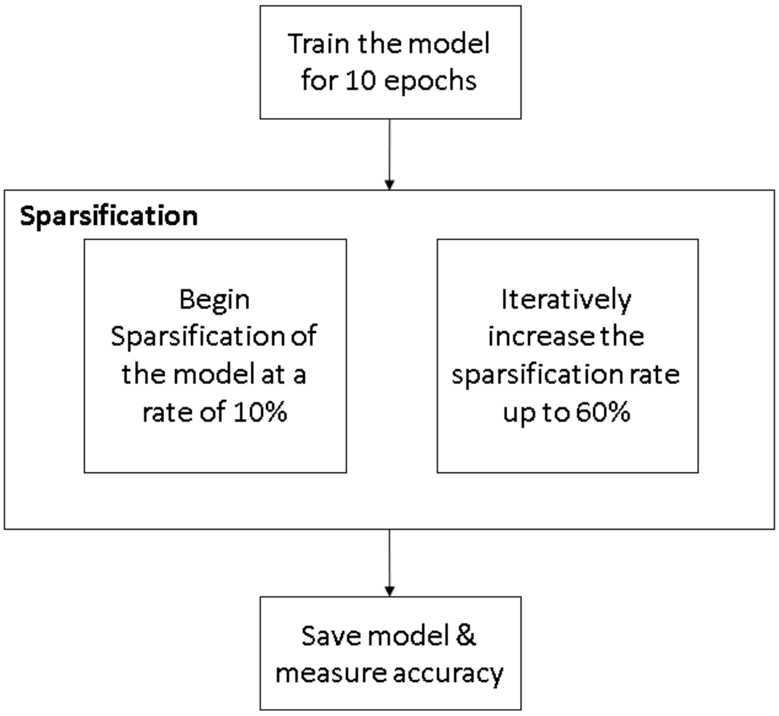
Sparsification is the process of repeatedly pruning the weights of the layer during the training process. The model is trained with a weight decay and if the model if the absolute weight of the model falls below a certain threshold, it is rounded to 0. Several papers have suggested using these techniques in Convolutional Neural Networks to improve the model performance in a contrained setting. (Bhattacharya et al.) considers including sparsification in their SparseSep system for improvement in model performance for making processors viable. Sun et al. uses sparsification in the model layers to reduce overfitting of the model. This sparsity can be structurally constrained to get a speed-up on GPUs which use matrix-multiplications to implement convolutions.

Sparsification of the network is performed during training. The weights of the network are allowed to settle (i.e when the rate of change of the training loss), is left to be constant.

Sparsification is enabled for each convolutional layer. We use the `prune_low_magnitude` function defined as part of `tensorflow_model_optimization.sparsity.keras` module.

```
x = Conv2D(filters=n_filters, kernel_size=(kernel_size, kernel_size), ksize),
kernel_initializer="he_normal",padding="same")
x =  sparsity.prune_low_magnitude(x, **prune_prm)
```

The `prune_prm` takes an instance of the `PolynomialDecay` class, which defines the pruning schedule for the layers.


Train the model for 10 epochs
Sparsification
Begin Sparsification of the model at a rate of 10%
Iteratively increase the sparsification rate up to 60%
Save model & measure accuracy

Quantization

Quantization is the process in which we can achieve a speedup in our computation by optimizing our model to perform integer operations instead of double precision floating point operations. In a typical GPU like K80, the number of floating point operations for the Double precision case is 1.87 Tflops which increases upto 5.6 Tflops for the Single precision case.

The resulting model is coverted into a TF-Lite model, which helps quantize it. Below is a code snippet to quantize a trained model:

```
# unet_inst.model is a pretrained unet model
converter = tf.lite.TFLiteConverter.from_keras_model(unet_inst.model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_quant_model = converter.convert()
```

Inference of the model is computed using an interpreter instance which takes an input tensor and produces the corresponding output tensor.

Training Setup and Hyper-parameters

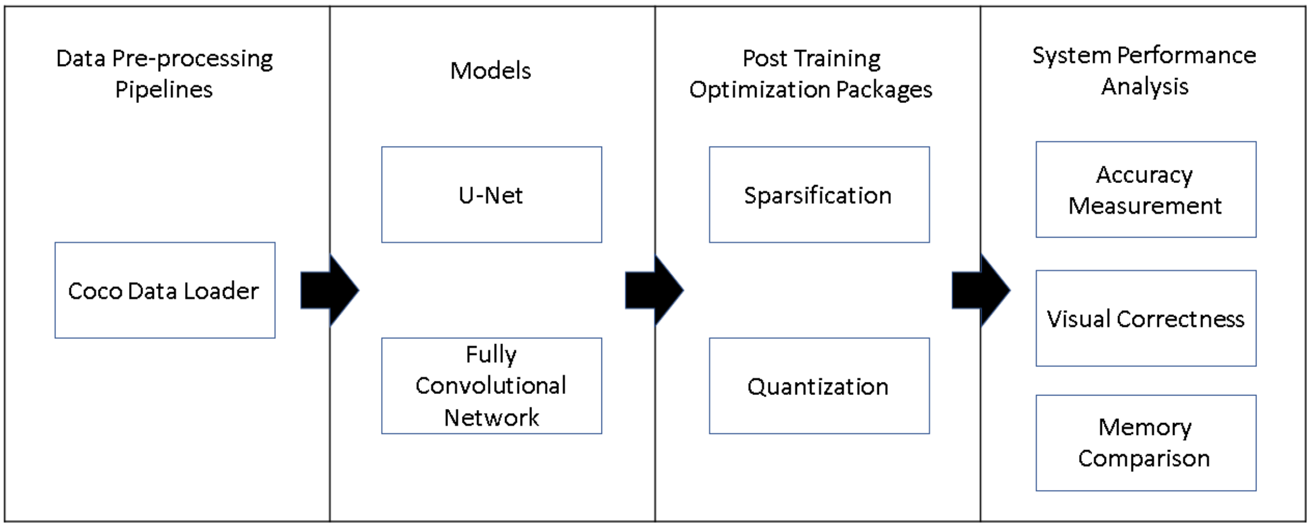
The COCO dataset consists of 117,266 images which is used for training. In addition 4952 images are used for validation. The network is trained from scratch with the following parameters:

- Batch Size: 40
- # Epochs: 20
- Dropout: 0.05
- Kernel Size: 3
- Batch Normalization: True
- # Filters: 16 -- # filters for the first layer, grows by a factor for successive layers
- Initial Sparsification ratio: 0.0 -- Initial sparsification is assumed to be 0
- Final Sparsification ration: 0.60 -- Set final sparsification to 0.60
- Initial Sparse Step: 30000 -- Begin sparsification after 10 epochs
- Final Sparse Step: 60000 -- Finish sparsification after 10 more epochs
- Sparse Frequency: 100 -- Sparsify after every 100 mini-batches

System Architecture

To build and compare models at scale, we required a modular approach to train, compare, sparsify/quantize and infer the results of our deep learning models.

Below is the architecture we have open sourced at: https://github.com/MADHAVAN001/semantic-segmentation (https://github.com/MADHAVAN001/semantic-segmentation)

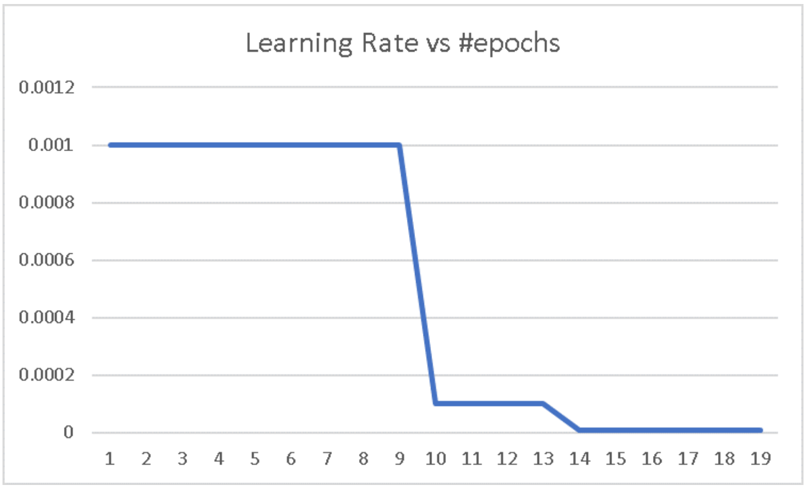

Data Pre-processing Pipelines
Coco Data Loader
Models
U-Net
Fully Convolutional Network
Post Training Optimization Packages
Sparsification
Quantization
System Performance Analysis
Accuracy Measurement
Visual Correctness
Memory Comparison

Results

All the results below are for the Fully Convolutional Network.

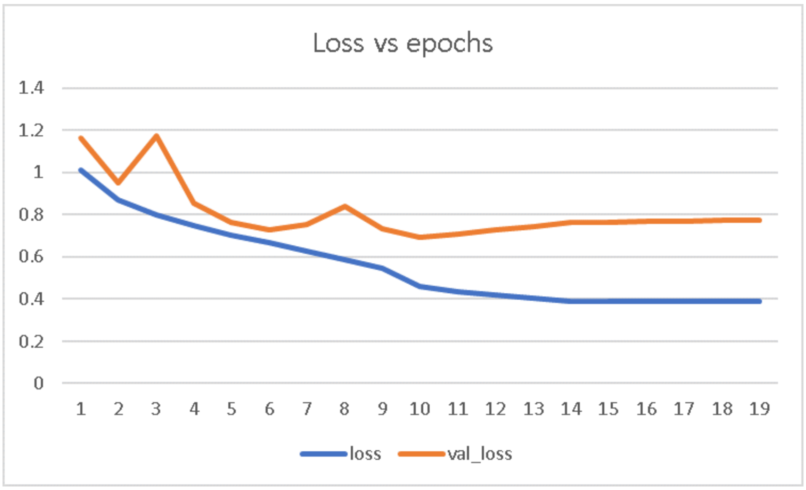
Learning Rate Scheduler

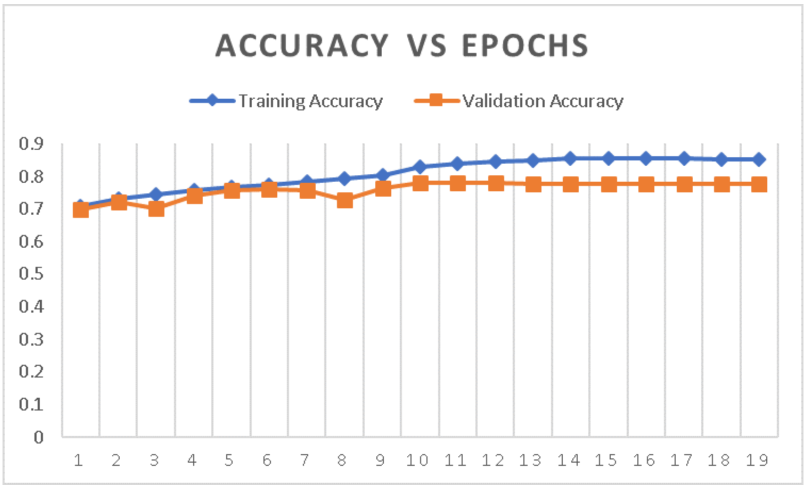
Starting with Adam optimizer for our loss function, we started with a decayed learning rate and this has been plotted in the graph below.



Accuracy and Loss over epochs

Running the training over multiple epochs, we recorded the training and validation accuracy, losses which are plotted in the two graphs below. The training accuracy started with 70% at the end of the first epoch and quickly stabilized to a value of 85% by the 15th epoch. In the case of validation accuracy, it also started with 70% but did not increase beyond 77% during our training procedure.

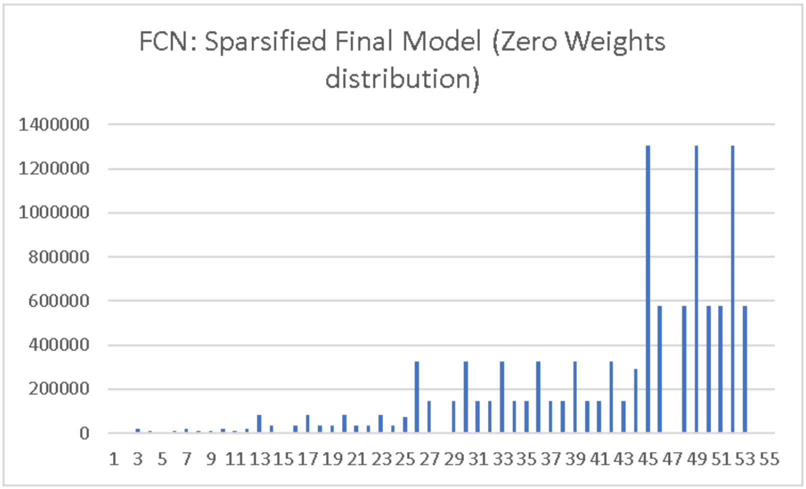




Sparsification

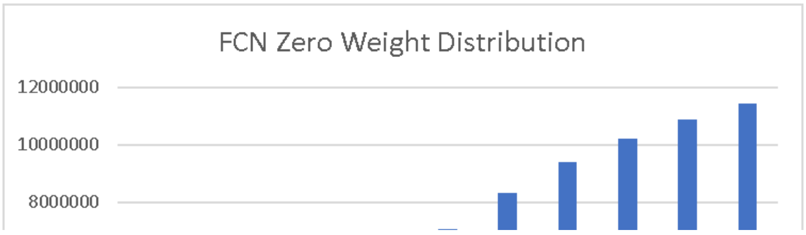
Zero Weight distributions over the classes

To ascertain that our sparsification was working over the Fully Convolutional Model, we plotted the distribution of zero weight distributions. It can be observed that the zero weight distributions are concentrated more towards the Dense layers as expected from the distribution of weights among the layers.



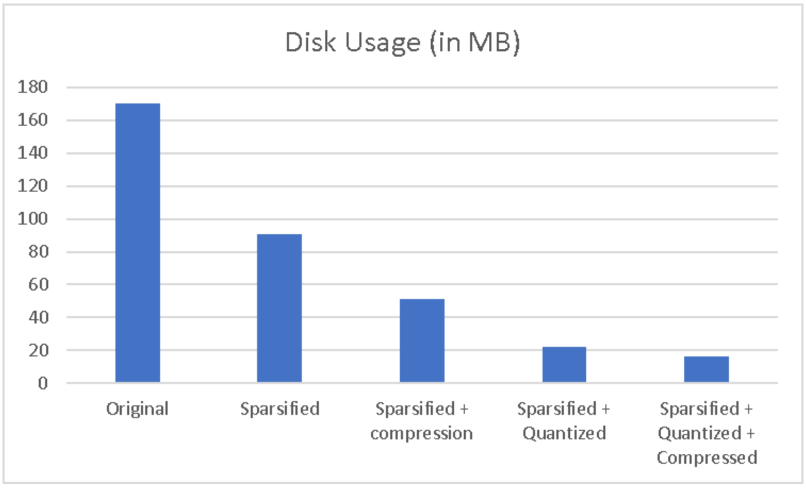
Number of Zero Weights per epoch (starting at 10th epoch)

We can also ascertain that iterative sparsification is working accurately by plotting the number of 0 weights after the end of each epoch starting at 10th epoch. We see that this increases from ~2 million in the 12th epoch to about 120 million in the 19th epoch.



Model Disk Usage

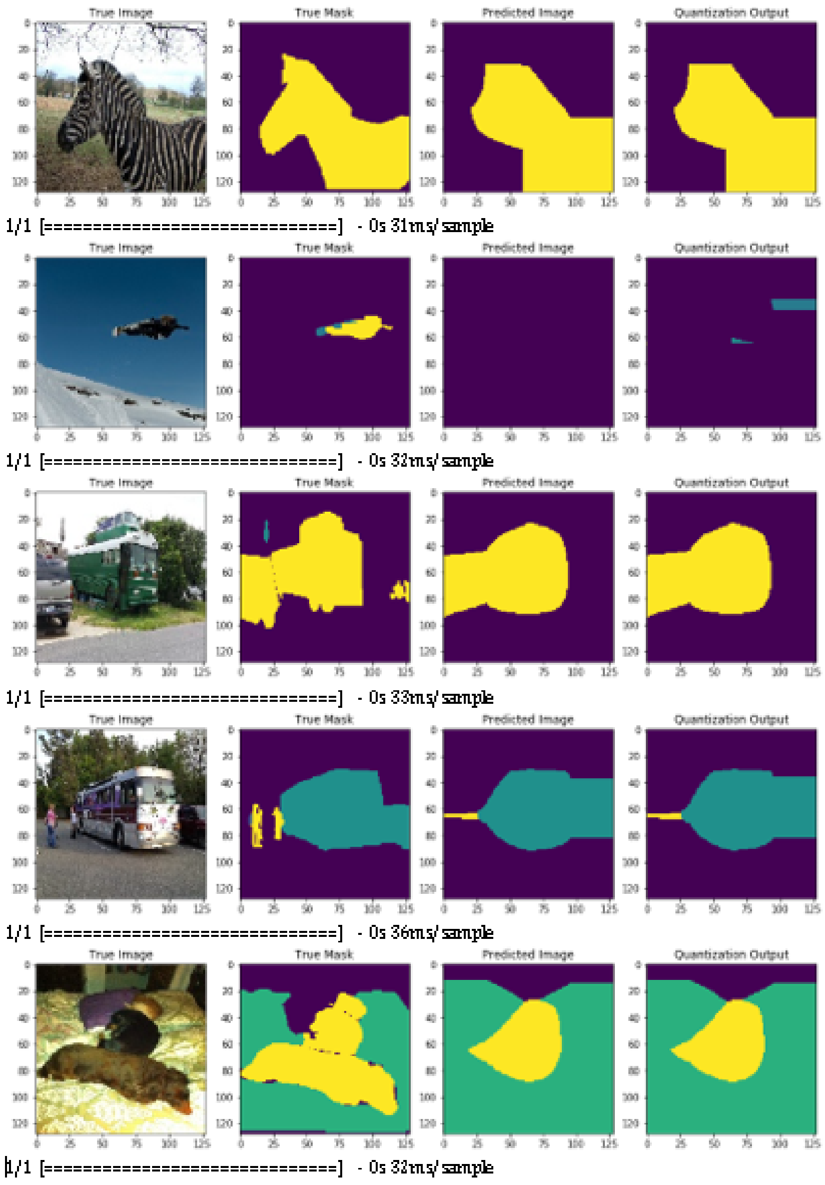
Disk utilization is an important metric for comparing the optimization performance of our models with the different techniques that we are using for improving our model. We see that from a size of ~170 MB for the original model, we can reduce it to ~18 MB after sparsification and pruning.



Model Accuracy

Given our optimization techniques are working as expected, we now need to ensure that visually our model's predictions do not vary. This analysis is done in the following few diagram below.

Without Sparsification

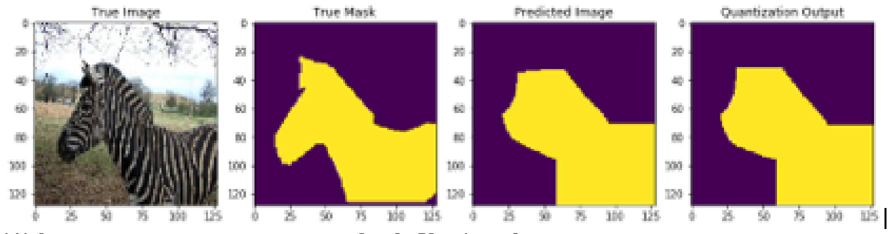


```
1/1 [==============================] - 0s 31ms/sample
1/1 [==============================] - 0s 32ms/sample
1/1 [==============================] - 0s 33ms/sample
1/1 [==============================] - 0s 36ms/sample
1/1 [==============================] - 0s 32ms/sample
```

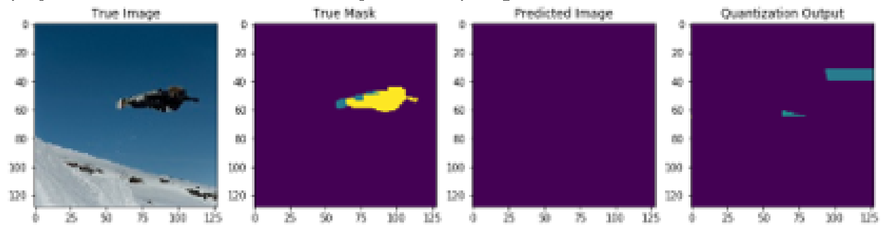
With Sparsification

Despite zeroing out many weights, the validation accuracy increases when compared to the non-sparsified model. The model after sparisification and quantization still performs well as there's no significant loss in segmentation accuracy.

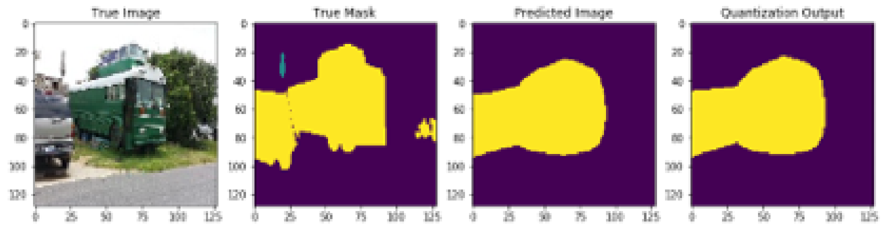
```
1/1 [==============================]  - 0s 32ms/sample
```



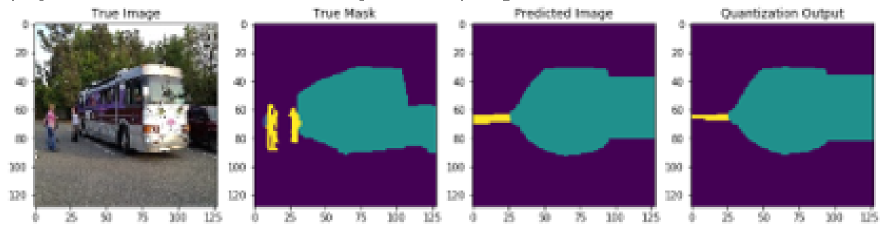
```
1/1 [==============================]  - 0s 32ms/sample
```



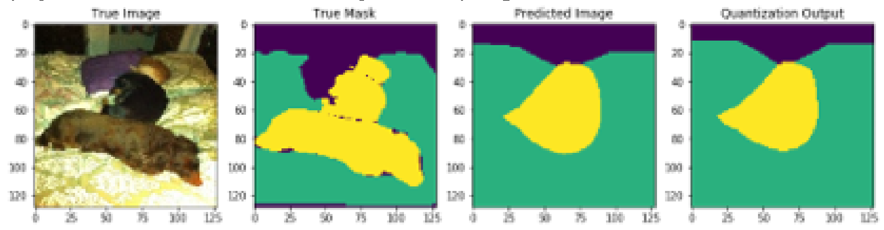
```
1/1 [==============================]  - 0s 32ms/sample
```



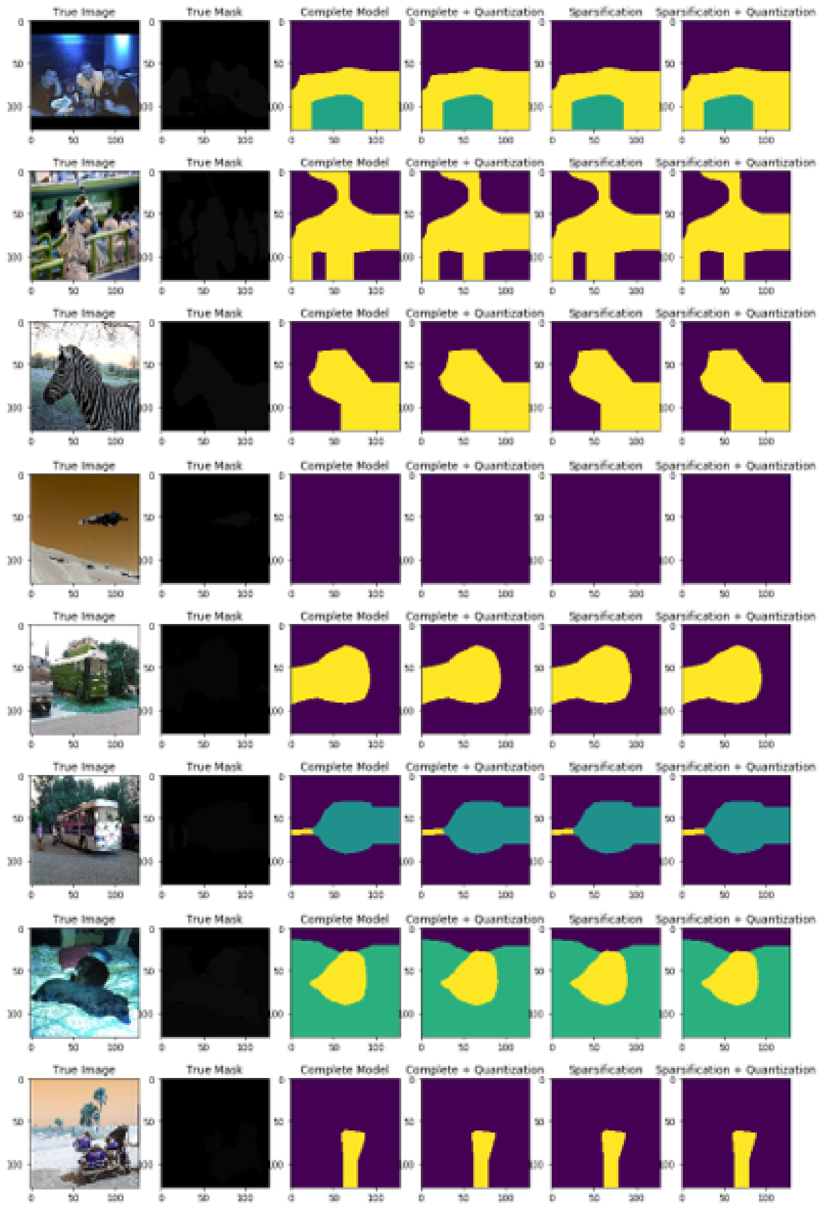
```
1/1 [==============================]  - 0s 32ms/sample
```



```
1/1 [==============================]  - 0s 32ms/sample
```



Complete Comparison



Limitations

The system architecture that we have proposed is meant to be scalable with the additions of models and datasets. But for the specific model which we have implemented, there are a few limitations. Image size is 128 X 128 which is smaller than thhe typical images used for the use cases mentioned in the use cases. With the increase in image size, there is expected to be an increased computational cost to the model. Specifically, there will be an addition of a convolution block to the model.

The current model doesn't prune layers with zeroed out weights. This isn't necessary for accelerators which convolve using matrix multiplications, however, the size of the model can be size of the model does significantly reduce with pruned layers. While existing packages on tensor-flow such as Keras-surgeon (Whetton et al.) were explored, compatibility issues have plagued its implementation.

Future Work

The number of computations in the model still largely remain the same as the pruned out weights are not removed from the network. As such, wven though the model is compressed using tf.lite, on running it on the processors, the original size difference can be increased by removing redundant kernels from the convolutional operations. L1 and L2 norm can be used to determine the most useful kernels for our task.

A teacher-student model can also be used, one which is similar to the implementation in Mask_RCNN in which we have a large set of labelled images from the Teacher model and we use a smaller Student model to perform as good as the teacher model. The student model is expected to be a much smaller version compared to the teacher model and a pairwise distillation is performed.

Conclusion

This project demonstrates a successful implementation of a simple Semantic Segmentation model such as a Fully Convolutional Network which was built in an architecture meant to provide the ground for comparison and analysis of a multiple models.

As such, another model UNet used in semantic segmentation was run by Shravan on the same dataset and a performance analysis and comparison was demonstrated in class presentation.

Our complete code is open sourced at https://github.com/MADHAVAN001/semantic-segmentation (https://github.com/MADHAVAN001/semantic-segmentation)

References

1. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556, 2014.
2. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.
3. P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to SIFT. CoRR, abs/1405.5769, 2014. 1
4. J. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In NIPS, 2014.
5. H. Li, K. Ota and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing," in IEEE Network, vol. 32, no. 1, pp. 96-101, Jan.-Feb. 2018. doi: 10.1109/MNET.2018.1700202
6. M. Hosseini, T. X. Tran, D. Pompili, K. Elisevich and H. Soltanian-Zadeh, "Deep Learning with Edge Computing for Localization of Epileptogenicity Using Multimodal rs-fMRI and EEG Big Data," 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, 2017, pp. 83-92.
7. M. N. Bojnordi and E. Ipek, "Memristive Boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), Barcelona, 2016, pp. 1-13.
8. C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 36, no. 3, pp. 513-517, March 2017.
9. Wang, Y., Shi, T., Yun, P., Tai, L., & Liu, M. (2018). Pointseg: Real-time semantic segmentation based on 3d lidar point cloud. arXiv preprint arXiv:1807.06288.
10. Treml, M., Arjona-Medina, J., Unterthiner, T., Durgesh, R., Friedmann, F., Schuberth, P., ... & Nessler, B. (2016, December). Speeding up semantic segmentation for autonomous driving. In MLITS, NIPS Workshop (Vol. 2, p. 7).
11. Ros, G., Sellart, L., Materzynska, J., Vazquez, D., & Lopez, A. M. (2016). The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3234-3243).
12. Jonathan Long, Evan Shelhamer, Trevor Darrell; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431-3440
13. Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick; The IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2961-2969
14. Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S., & Pal, C. (2016). The importance of skip connections in biomedical image segmentation. In Deep Learning and Data Labeling for Medical Applications (pp. 179-187). Springer, Cham.
15. Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.
16. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).
17. He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask r-cnn. In Proceedings of the IEEE international conference on computer vision (pp. 2961-2969).
18. Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, Hannaneh Hajishirzi; The European Conference on Computer Vision (ECCV), 2018, pp. 552-568
19. Gaetan Bahl, Lionel Daniel, Matthieu Moretti, Florent Lafarge; The IEEE International Conference on Computer Vision (ICCV), 2019, pp
20. Alom, M. Z., Hasan, M., Yakopcic, C., Taha, T. M., & Asari, V. K. (2018). Recurrent residual convolutional neural network based on u-net (R2U-net) for medical image segmentation. arXiv preprint arXiv:1802.06955.

21. Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 658-666).
22. Bhattacharya, S., & Lane, N. D. (2016, November). Sparsification and separation of deep learning layers for constrained resource inference on wearables. In Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM (pp. 176-189). ACM.
23. Sun, X., Ren, X., Ma, S., & Wang, H. (2017, August). meprop: Sparsified back propagation for accelerated deep learning with reduced overfitting. In Proceedings of the 34th International Conference on Machine Learning-Volume 70 (pp. 3299-3308). JMLR. org.
24. Ben Whetton -- Keras Surgeon. https://github.com/BenWhetton/keras-surgeon (https://github.com/BenWhetton/keras-surgeon)