Semantic Segmentation - Milestone 2

Monday, April 13, 2020 2:46 AM

Team: Nikita Goswami and Vimal Kumarasamy

Project progress

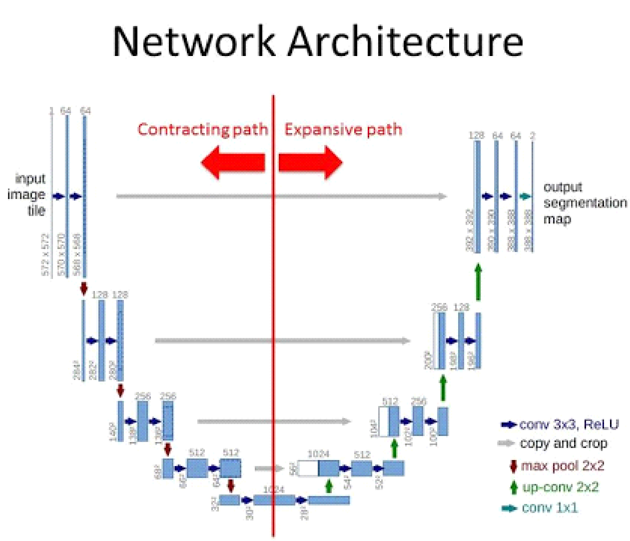
- Transfer learning
- Building U-net
- Dataset
- Training
- Comparison of performance across models
    - Experiment 1
    - Experiment 2
- Utility functions
- Additional pointers

Transfer Learning

- The features such as edges, corners, local shapes, patterns, textures can be learned from images with the help of kernels, however training one from scratch might take time and effort
- There are available frameworks such as Mobile net architectures, which has been trained on lot of images and those architectures know what to expect in an image
- Such an architecture is readily available in TensorFlow - MobileNetV2
- MobileNetV2 has been leverage which takes an input of size (128,128,3)
- The kernels (weights) from MobileNetV2 are used to convert the raw pixel intensities into features, however these weights will not be trained during the learning process

Building U-Net

- While handling any neural network with deeper layers, there optimization process depends on the error / delta from the subsequent layers
- While performing backward propagation, if the delta values are not significant enough for the inner layers those layers might not be getting changed, resulting in a condition called as vanishing gradients
- These cases can be avoided with the help of skip layers
- Skip layers are those when the throughput from a layer can be sent to subsequent layers by skipping the immediate next layer
- One such case is called as U-nets, which is being employed in our case of semantic segmentation
- The below image is for illustration



- A U-net architecture with 4 layer down stack and 4 layer up stack has been used, with skip connections
- These layers will be taking the output from the pretrained weights from the mobilenetv2 network, however these weights would be trainable
- The final network summary is shown below

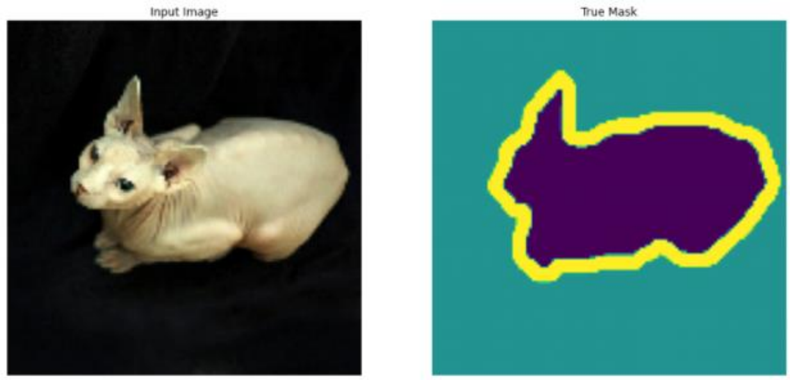
```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_16 (InputLayer)           [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
model_10 (Model)                [(None, 64, 64, 96), 1841984     input_16[0][0]
__________________________________________________________________________________________________
sequential_32 (Sequential)      (None, 8, 8, 512)    1476608     model_10[1][4]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 8, 8, 1088)   0           sequential_32[0][0]
                                                                 model_10[1][3]
__________________________________________________________________________________________________
sequential_33 (Sequential)      (None, 16, 16, 256)  2507776     concatenate_8[0][0]
__________________________________________________________________________________________________
concatenate_9 (Concatenate)     (None, 16, 16, 448)  0           sequential_33[0][0]
                                                                 model_10[1][2]
__________________________________________________________________________________________________
sequential_34 (Sequential)      (None, 32, 32, 128)  516608      concatenate_9[0][0]
__________________________________________________________________________________________________
concatenate_10 (Concatenate)    (None, 32, 32, 272)  0           sequential_34[0][0]
                                                                 model_10[1][1]
__________________________________________________________________________________________________
sequential_35 (Sequential)      (None, 64, 64, 64)   156928      concatenate_10[0][0]
__________________________________________________________________________________________________
concatenate_11 (Concatenate)    (None, 64, 64, 160)  0           sequential_35[0][0]
                                                                 model_10[1][0]
__________________________________________________________________________________________________
conv2d_transpose_38 (Conv2DTran (None, 128, 128, 3)  4323        concatenate_11[0][0]
==================================================================================================
Total params: 6,504,227
Trainable params: 4,660,323
Non-trainable params: 1,843,904
__________________________________________________________________________________________________
```

Experimenting with resolutions

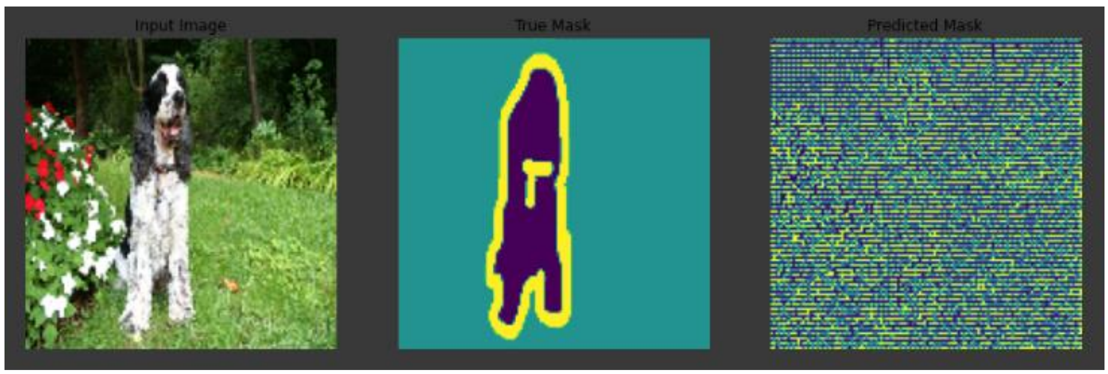
- The pretrained weights from Mobilenet can accept different resolutions of the input image
- As the raw input is far bigger than the compatible size in Mobilenet, bilinear interpolation based resizing has been performed
- While experimenting with different resolutions of the input images, interesting findings have been observed

Dataset

- In order to test the performance of the network, oxford IIIT pet dataset has been used 0 containing 37 categories of pet with 200 images per pet class
- The annotations are such that the main body of the pet, the boundary of the pet and the rest of the image are annotated as 3 different classes
- A batch size of 64 images have been chosen, and different operations such as shuffling the train images (along with the label) and the flipping randomly are employed
- The prediction will be made for every pixel, and the output layer is a deconvolutional, which will return an array of length 3, which is equal to the number of classes that we have started with
- Here is a glimpse of the dataset

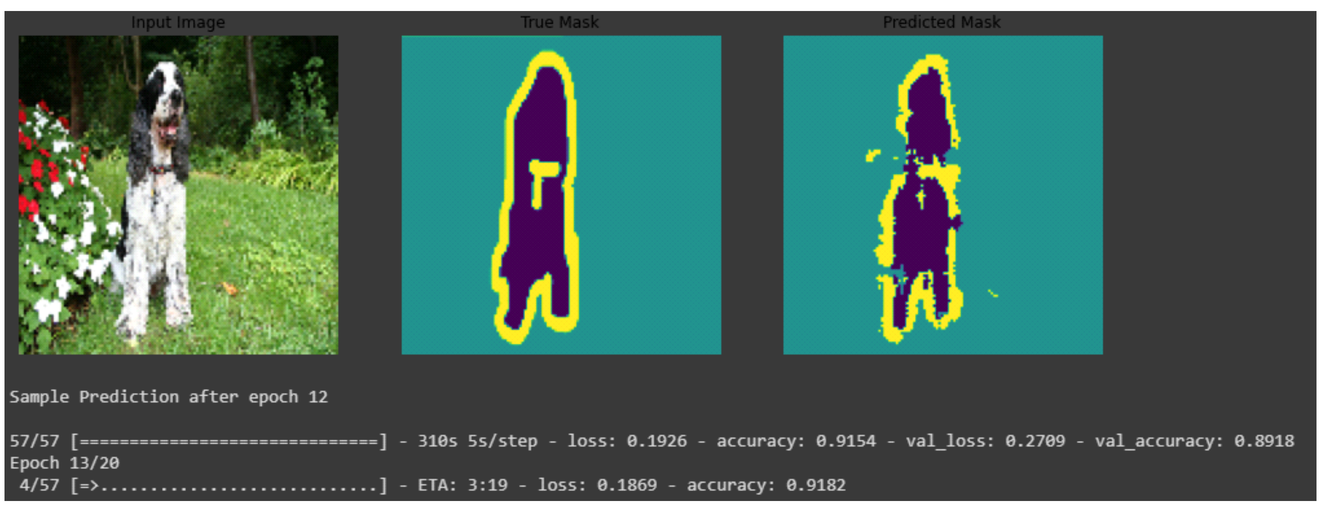


- Once the network is put together along with the dataset, the prediction without any training is done, which is shown here

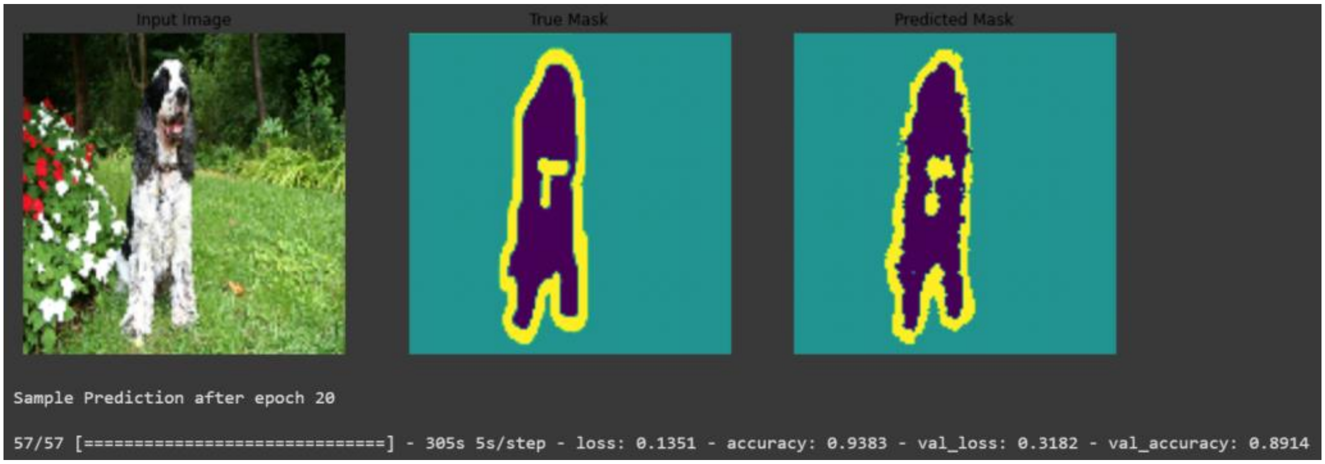


<u>Training</u>

- Across multiple training epochs, the same example was used for prediction and the improvement in the prediction can be observed here for 128 x 128 size input images



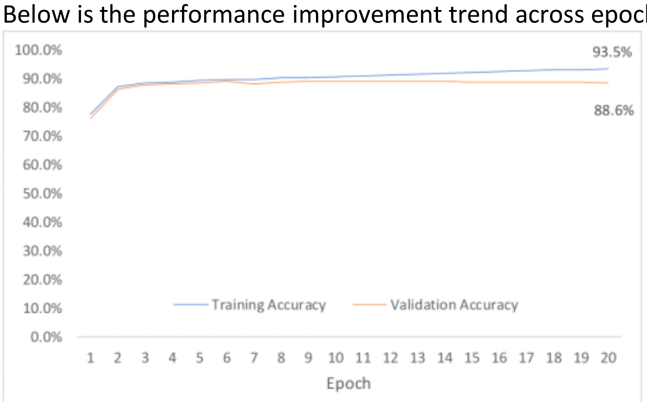
- After 20 epochs the prediction was much better than the initial prediction, which is shown here



Comparison of performance across models

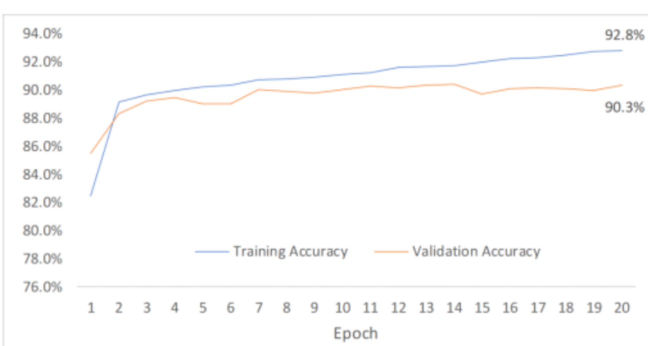
Experiment 1

- Below are the results from the model that accepted the input image of resolution 128 x 128
- The output format would be an image with the classes laid out next to the true image
- The model has attained an accuracy of 88.6% on the Validation dataset, and as expected its higher in training dataset - 93.5%
- Below is the performance improvement trend across epochs



Experiment 2

- Below are the results from the model that accepted the input image of resolution 224 x 224
- The performance summary is shown below



- The best training accuracy after 20 epochs has been observed as 92.8%, and the respective validation accuracy in 90.3%

Inference

- Enabling the architecture to process higher resolution images resulted in reduced overfitting
- With higher resolution, the training accuracy was lower compared to 128 resolution architecture,

however on the validation dataset the model with higher resolution performed much better with an accuracy of 90.3%

- This is an indication of overfitting that has been avoided by introducing higher variance in the dataset with the help of higher resolution images

<u>Utility functions</u>

- To enable easier prediction and model fetching, the below utility functions are built
- Saving checkpoints: After every successful epoch the checkpoints are saved in a location (provided by the user)
- These checkpoints will be used when the user wants to predict the classes on a new dataset
- Upload the image: The users are also enabled to directly upload an image on the notebook and the prediction is done
- Unet building: A function that builds unet framework on top of the given mobilenet pre-trained weights
- Display image: At multiple instances, to check the quality of the prediction a function to showcase the prediction has been built
- Callbacks: When a user preferred accuracy level has been met, the function can terminate further training the checkpoints will be saved, this is also has been added

<u>Additional pointers</u>

- CamVid dataset has more than 30 classes, and those classes can be reduced to fit into the problem at hand
- However in order to learn semantics in a real world traffic situation, a network that can scale the image faster would be required, such as Atrous layers which might be required
- The next step would be to make the architecture compatible with any pretrained weights and perform prediction
- The entire project has been documented on