**Road Extraction from Satellite imageries**

This repository contains code to train unet model with efficientNet backbone to segment road surface from satellite imageries. It includes, code for :

* Data cleaning
* Data pre-processing
* Model training
* Model inference
* Inference post-processing
* Model deployment using tensorflow-serving

**Installation:**

* Require - Python 3.6 or above
* To install - pip install -r requirements.txt

**Dataset:**

* Training Dataset: 1105 images and 804 labelled mask
* Test Dataset: 13 images and 13 labelled mask

**Pre-processing:**

* The training dataset contains 1105 images of size (1500\*1500) but for only 804 images we had the corresponding labelled mask. So only images having corresponding masks were utilized for training.
* Out of 804 images with mask, there are images with white patches in them but have labelled data for those white patch regions. Such images diminish model performance, and therefore were not used during training (if in masks, for those pixels overlapping with white patches were not labelled than we could have used these images too). One such example is shared below:

A picture containing text

Description automatically generated A picture containing shape

Description automatically generated

Image Mask

* Each of the remaining images and masks were then resized to (1536\*1536) and then broken into nine images of size (512\*512). Benefits gained from splitting images are:
  + Larger training dataset and more options for augmentation. Each of the nine images can have different augmentation at run time which reduces the chance for overfitting.
  + Bigger batch size as more images of smaller size can be loaded into limited GPU memory as compared to larger images.
* Few more random crops of size (512\*512) from images of size (1536\*1536) were also taken to increase the dataset. These random crops do not overlap with nine cropped images to avoid data duplication. These images were randomly rotated by either 90 degrees or 270 degrees.
* After preprocessing, total of 7240 images were obtained for training.
* All 13 images from testing set were correct and used directly during model performance evaluation.

**Training:**

* Training set was divided in 85:15 ratio, to obtain 6150 training images and 1090 validation images.
* Tensorflow v2 along with tensorflow keras is used to build model.
* Augmentation:
  + Runtime augmentation was performed on training dataset to increase dataset variety.
  + Combination of horizontal and vertical flip with a probability of 0.5.
  + Random hue, delta and saturation.
  + Brightness augmentation to make model deal better in low light situations.
* Tensorflow dataset api is used to pre-process data before feeding it into model. Multiple cpu cores were enabled to prevent cpu bottleneck.
* **Model Architecture:**
  + Model is build on Unet architecture with Efficientnetb4 as backbone. Unets were originally build for medical purposes to segment small objects from large images. This capability makes unet a good candidate to be used in satellite imageries segmentation problems.
* **Loss Functions:** 
  + A custom **boundary loss function** was created to calculate loss at road boundaries. Main objective of this loss function was to get smoother outputs at road boundaries. This also improves the model output in case of two very close parallel roads.
  + Boundary loss function was used in combination with **BCE-Dice Loss.**
* **Training schedule:**
  + Model was trained for first 10 epochs with combination of boundary loss and bce-dice loss. And further finetuned for another 30 epochs with using only bce-dice loss.
  + Cyclic learning rate was used with cycle size of 5 epochs and leaning rate decay of 0.8 (20 percent) after each cycle.
  + Weights were saved after every epochs only if validation loss is lower than previous epoch. This is to ensure that only best weights are saved and used during inference.
  + Due to early stopping callback, training stopped at epoch 30. Best weights (lowest validation loss) were saved at step 24. More detailed explanation is provided later in the document under callbacks section.
  + Weights were also saved after each cycle and can be used for Polyak-Ruppert averaging (weighted averaging of models weight).

**Callbacks:**

Various callbacks were created to perform various actions during training.

* **EarlyCheckpoint:** This preventsmodel overfittingby stopping training early. During training, validation loss was calculated after every epoch and if the loss has not decreased for certain continues epochs (10 epochs or two learning rate cycle was used) and the training is stopped.
* **ModelCheckpoint:** It saves the weights after every epoch/batches. This was configured to save only the best weights (i.e. only if validation loss is lower than all the previous epochs).
* **Tensorboard Image:** It is used to save various training parameters for visualization. It was configured to save loss, recall, precision, lr graphs for both training and validation dataset. Random samples of ten validations images along with their mask and prediction is also saved to visualize training performance along with model graph.
* **Callback to save weights at lr cycle end:** This saves weights after every learning rate cycle. This weights can be used for Polyak-Ruppert averaging. Generally loss is minimum after every lr cycle, which signifies that model achieved a local minima. So averaging out these weights can give better results as compared to last saved weights.

**Training visualization:**

Following graphs shows training progress:

* Red line implies validation data
* Orange line implies training data

First 10 epochs (with both boundary loss and bce-dice loss):

**Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated Chart, line chart

Description automatically generated**

Learning rate Epoch loss Epoch iou

Training For next 30 epochs (with bce-dice loss only):

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated Chart, line chart

Description automatically generated

Learning rate Epoch loss Epoch iou

Tensorboard visualization example for validation samples:

**A picture containing text, plant

Description automatically generated Shape

Description automatically generated Shape

Description automatically generated**

**Image Mask Prediction**

**Inference:**

For inference on test images, each images were divided into (512\*512), similar to the training preprocessing and the model prediction is then stitched together to produce predicted mask of size (1500 \* 1500).

* **Post processing:** 
  + To get binary image from prediction output, thresholding of 0.5 applied on each mask. Any pixel with value above 0.5 were considered to be positive road pixel.
  + Small blobs (white patches) of false positives were removed.

**Model Evaluation:**

Model is evaluated on 13 test images and following parameters were calculated:

* Intersection over union
* Precision
* Recall
* F1-score
* Accuracy

For each image, all the evaluation is done on pixel level where each pixel is designated either true positive, false positive, true negative or false negative.

Average for each parameters was calculated, for all 13 images to get mean values.

Mean values of accuracy parameters are higher on test dataset for best weights as compared to last weights saved. Also because of early stopping, model didn’t train for much longer once model started to overfit.

Table below shows the parameters values after various stages of training:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Stages | Mean\_iou | Mean\_precision | Mean\_recall | Mean\_F1 |
| Epoch - 10 | 0.6041 | 0.7204 | 0.7907 | 0.7521 |
| Epoch - 30 (Final weights) | 0.6067 | 0.7275 | 0.7862 | 0.7534 |
| Epoch – 24 (Best weights) | 0.6171 | 0.7315 | 0.7975 | 0.7612 |

**Deployment**

Tensorflow serving is used to deploy the model in production. Some of the benefits of using tf-serving are:

* Support for both rest calls or gRPC calls.
* Out of the box model management and inference apis.
* Support multiple model per server/instance.
* Support multiple versions of each models.
* Easy to scale.

**Further Developments:**

**Post-processing:**

* Skeletonization techniques can be implemented to extract road centerline from road surface.
* Scikit image (skimage.segmentation.find\_boundaries) can be used to extract boundaries from output.

**Improvements:**

* Model can be trained on bit bigger images. For example (splitting original image into 4 (750\*750) images. Bigger images will have less cases where roads are at boundaries as compared to smaller images, which will result in better context around more roads.
* Instead of bce-dice loss during finetuning, focal loss can be used. Using focal loss would have made model learn better on cases where it is performing very bad.
* Various other augmentation like partial rotation, distortion etc could have been used.
* Various other model architecture can be trained like **D-linkNet**. It is a state of the art network to extract line features from satellite images.