Parking Spot Recognition Solution(s)

The parking spot recognition solution(s) are required to identify vacancies or occupancy in the parking spaces using Geospatial Imagery Analytics methods. It utilizes Computer Vision, and training Machine Learning models, to extract open parking spots in a parking space based on data extraction from video feeds from the security cameras.

The recognition solution(s) will be used for object detection to identify all of the parked vehicles and to verify overlapping of the vehicles with the respective parking spot. Convolutional Neural Networks (CNNs), that fall under the domain of Deep Learning Scheme for Computer Vision, were used to analyze each parking spot and to predict occupancy or vacancy of those spots. The vehicles and objects of interest are located and classified within certain boundaries. Deep neural networks differentiate the vehicles from objects that are not vehicles by analyzing a multitude of features. Appropriate artificial intelligence systems or machine learning models were developed as best fit for the solution, and were trained using the video feed data. Image segmentation and other techniques and tasks were undertaken for refinement (as required), to achieve a satisfactory accuracy of spot vacancy/occupancy detection with the model. The phases include detection and IoU(Intersection Over Union). Detection decisions are returned as Boolean. Research and some experimentation was required with the existing popular AI models to compare results and to identify the best-fitting model for the solution; Fine tuning and modifications were undertaken.

For the Parking Spot Recognition solution(s), deep learning architecture in CV: VGG-16 was used for one of two solutions, while the second solution was based on a custom 3X64 Node Convolutional Neural Network model. The VGG16 model trains on RGB color image dataset and the 3X64CNN model on grayscale data upon preprocessing; One model can yield better accuracy/loss than the other, depending on the type/quality/amount of data we are able to train with (considering the limited resources during the crisis timeframe). Both models implements the following:

Once a dataset is populated for training the model, it first undergoes Pre-processing where the images are normalized, converted, etc. to prepare it to be fed into training the model. Data augmentation could be done to improve the datasets if there is a lack of data in this timeframe and deal with overfitting to some extent. The validation sets and test sets were prepared as well and directories set-up accordingly.

The models were then built. The VGG-16 model was fine-tuned using eg. Keras (a NN-library and API for building and training deep learning models) and the CNN model was tweaked to fit the project solution. Next, the models were trained using the pre-processed datasets. The epoch value and other parameters can be tweaked for optimizations.

Predictions were run and statistical tools such as confusion matrix were plotted to assess the outcome of factors such accuracy and losses. The models were tweaked and fine-tuned as necessary, and were re-trained. The 3X64-CNN model, during a session, yielded >97% accuracy on the test set which could be. The prediction outputs along with the spot locators or indexed locations were returned as JSON and saved. Endpoints were established so that these output objects can be transmitted and utilized by the front-end.

The model was saved as a .h5 file and was deployed on GeoEvents-ArcGIS Server so that calls can be received by the front-end of web-application at the prediction endpoint.

The SpotNN_vgg16 model code includes detailed comments, including some documentations and some important scripts/commands that might be very useful for future developers to work

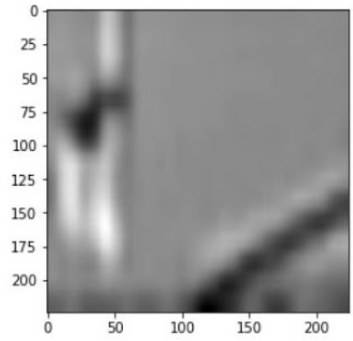
on or improve on this proof-of-concept solution, and makes the code very easy to follow. In addition, a demo front-end and Flask web services were used to pseudo-deploy/host and test the CV solution components/models prior to pushing it to github/project.

Some of the technologies, libraries, and/or platforms include OpenCV, Tensorflow, Matplotlib, numPy, keras, skLearn, Python, and Project Jupyter. Google CoLab was primarily used to avoid dependency and many other issues that arose initially. Tools and technologies found via research were used for the project as needed. Polygon selection tools, spot extractors, and other tools developed by the team supplements the technological needs for this project.

The scope and features of the recognition solution was dependent on the time constraint, access to, and availability of resources during the crisis. Parking lots are often empty and have a limited variety of parked vehicles and hence the dataset to train the model was limited. The model could benefit from image data extracted with different weather patterns, shadows, time of days, variety and size of vehicles, and so on, however, the model can simply be re-trained with a better dataset in the future in order to get better predictions on a larger variety of instances. The solution(s) are aimed to demonstrate a proof-of-concept as a deliverable; Additional features or capabilities can be added later or improved, with sponsors and mentors approval and given additional time and resources.

A screenshot of the run incl. accuracy/losses stats is provided below:

```
(30, 56)
```




```
381
0
1
0
1
1
1
1
1
1
1
Epoch 1/10
WARNING:tensorflow:Layer conv2d_39 is casting an input tensor from dtype float64 to the layer's dtype of f
loat32, which is new behavior in TensorFlow 2.  The layer has dtype float32 because it's dtype defaults to
floatx.

If you intended to run this layer in float32, you can safely ignore this warning. If in doubt, this warnin
g is likely only an issue if you are porting a TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call `tf.keras.backend.set_floatx('float64')`. To c
hange just this layer, pass dtype='float64' to the layer constructor. If you are the author of this layer,
you can disable autocasting by passing autocast=False to the base Layer constructor.

11/11 [==============================] - 13s 1s/step - loss: 4.9211 - accuracy: 0.6901 - val_loss: 0.5901
- val_accuracy: 0.8205
Epoch 2/10
11/11 [==============================] - 13s 1s/step - loss: 0.5146 - accuracy: 0.7456 - val_loss: 0.3189
- val_accuracy: 0.7692
Epoch 3/10
11/11 [==============================] - 13s 1s/step - loss: 0.3393 - accuracy: 0.8304 - val_loss: 0.2174
- val_accuracy: 0.8718
Epoch 4/10
11/11 [==============================] - 13s 1s/step - loss: 0.3014 - accuracy: 0.8450 - val_loss: 0.2054
- val_accuracy: 0.8974
Epoch 5/10
11/11 [==============================] - 13s 1s/step - loss: 0.2454 - accuracy: 0.9123 - val_loss: 0.1825
- val_accuracy: 0.9231
Epoch 6/10
11/11 [==============================] - 13s 1s/step - loss: 0.2247 - accuracy: 0.9591 - val_loss: 0.1587
- val_accuracy: 1.0000
Epoch 7/10
11/11 [==============================] - 13s 1s/step - loss: 0.2216 - accuracy: 0.9503 - val_loss: 0.1398
- val_accuracy: 0.9744
Epoch 8/10
11/11 [==============================] - 13s 1s/step - loss: 0.2031 - accuracy: 0.9678 - val_loss: 0.1476
- val_accuracy: 0.9744
Epoch 9/10
11/11 [==============================] - 13s 1s/step - loss: 0.2081 - accuracy: 0.9708 - val_loss: 0.1489
- val_accuracy: 1.0000
Epoch 10/10
11/11 [==============================] - 13s 1s/step - loss: 0.1995 - accuracy: 0.9678 - val_loss: 0.1561
- val_accuracy: 0.9744
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource_variable_op
s.py:1817: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with c
onstraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: 64X3-CNN.model/assets
```