CG_TEAM_IGZYD

Problem Description:

In this hackathon, your mission, if you choose to accept is to solve a very critical problem in the overall Smart Parking solution - **"Identify available parking spots from parking lot camera images".**

The main usage of this project is to develop a Smart Parking system that typically obtains information about available parking spaces in a particular geographic area and process is real-time to place vehicles at available parking spots.

The main advantage of implementing this project is by deployed as a system, smart parking thus reduces car emissions in urban centers by reducing the need for people to needlessly circle city blocks searching for parking. It also permits cities to carefully manage their parking supply and finally, it reduces the daily stress associated with parking woes.

Dataset and its description:

There is one standard dataset for his problem called PK LOT but a new dataset was released in 2015 called CNR Parking and I preferred this dataset over PK LOT. The dataset is divided into 3 parts

1. A complete parking lot image of 1000*750 pixels image
2. A 150*150 pixels patched image of the individual parking lots
3. A CSV file for each camera to let us know the positions of each parking lot in that camera region.

There are a total of 9 cameras and I preferred to choose a camera for my problem because it is both simple and covers all the features. Here is the example image of a snap from camera 5

There are around 46 parking lots in this region and the CSV file is as follows

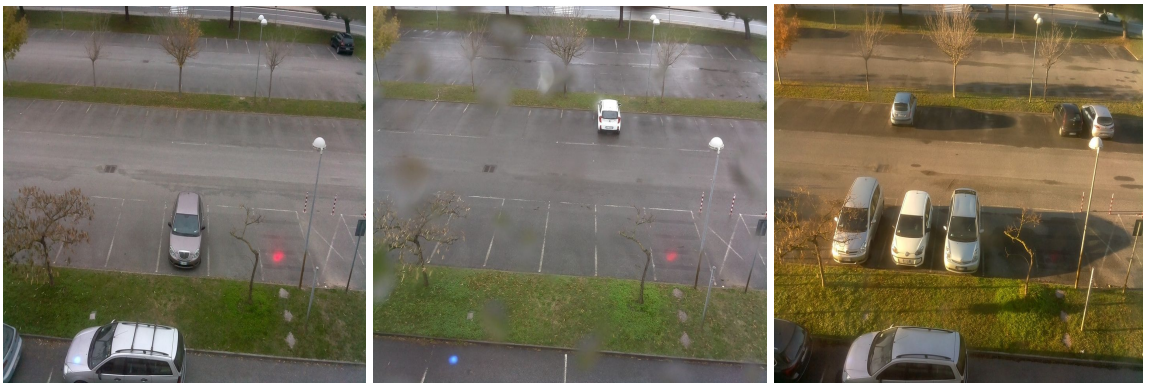
SlotId	X	Y	W	H
609	564	1594	380	349
197	1876	1046	300	300
198	1513	1053	330	330
199	1146	1004	360	360
200	794	1016	360	360
201	454	972	360	360
202	86	948	360	360
228	2210	508	216	216
229	1986	494	216	216
230	1732	496	216	216
231	1508	486	216	216

Where SlotID defines the unique ID for each parking lot, X, Y defines the left top point of the parking lot and the W, H defines the height width of the parking lot.

The data collected camera 5 is itself divided into 3 types based on weather conditions

1. Overcast
2. Rainy
3. Sunny

The sample images are shown below



Overcast ***Rainy*** ***Sunny***

Data Preprocessing:

Since the given images are 1000*750 a mask is applied to get the patches of the individual parking lot. These images are converted to a 224 * 224 pixels image and its purpose is explained in the next phase. After getting the image we use various preprocessing available from the **kears.preprocessing** like ImageDataGenerator to do operations like,
Rotation, width_shift, height_shift, shearing, zooming, horizontal_flip, vertical_flip etc.,

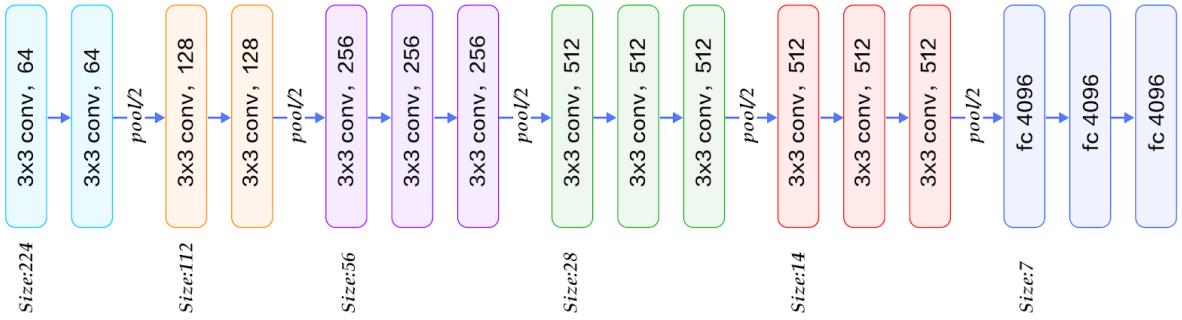
All these techniques are performed to make the model understand the features even if there is a change in the orientation of image during the testing phase.

There is one more phase in data preprocessing where I split the data as the Training set (named as A)and the Validation Set(named as B). There are around 2000 in both the splits. The trained set is used for training of my model and the validation set is used to verify my model. So this is the end of the data-preprocessing phase next follows the model selection and training.

Model Selection:

Convolution neural networks may look like a good model for his problem but to solve a problem like this lot of training data, time, Resource has to be spent on it. To prevent that I used Visual Geometry Group and it's famous as VGG in short. To let know about VGG, this model won the famous LeNet challenge in 2014.

Though there are various other models like RESNET and INCEPTION that are more advanced that VGG, VGG outperformed all of them. The only reason for the success of VGG is its architecture. The architecture of VGG is as follows



Initially, there are 2 Convolutional layers followed by a polling layer and convolutions layers followed by a polling layer. After that, there are 3 Convolutional layers followed by a polling layer and 3 Convolutional layers followed by a polling layer. Then at the end, there are 3 fully connected layers. This is the architecture of **VGG16** model, it's is trained on multiclass classification task with nearly 1000 classes.

For this problem, we use this VGG model but in the form of **Transfer Learning**.Transfer learning is something where we keep the weights of the network as constant and we replace the fully connected with a new classifier. The no of classes in our model defines the classifier we choose. For classifying 2 classes we use Binary Cross Entropy as the classifier and for the multi-class classification problem, we use Categorical Cross Entropy. The activation function for the former one is Sigmoid and for the latter one is Softmax.

VGG model is trained on images of size 224 * 224 pixels, this is the reason why we use input size as 244*244. During the training process it's our decision to whether to update or not to update the weights of the inner layer. Since the VGG model is already trained on thousands of images I preferred not to update the weights of inner layers and it's a time and resource consuming process.

For the initialization of the model without the last fully-connected layer, the following code is implemented

```
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet',
include_top=False)
```

To prevent the inner node form training the following lines of code is implemented

```
for layer in vgg.layers:
  layer.trainable = False
```

Then we train the model on the training data which is a collection of images in the form of patches. The sample patches are shown below

Busy Slot

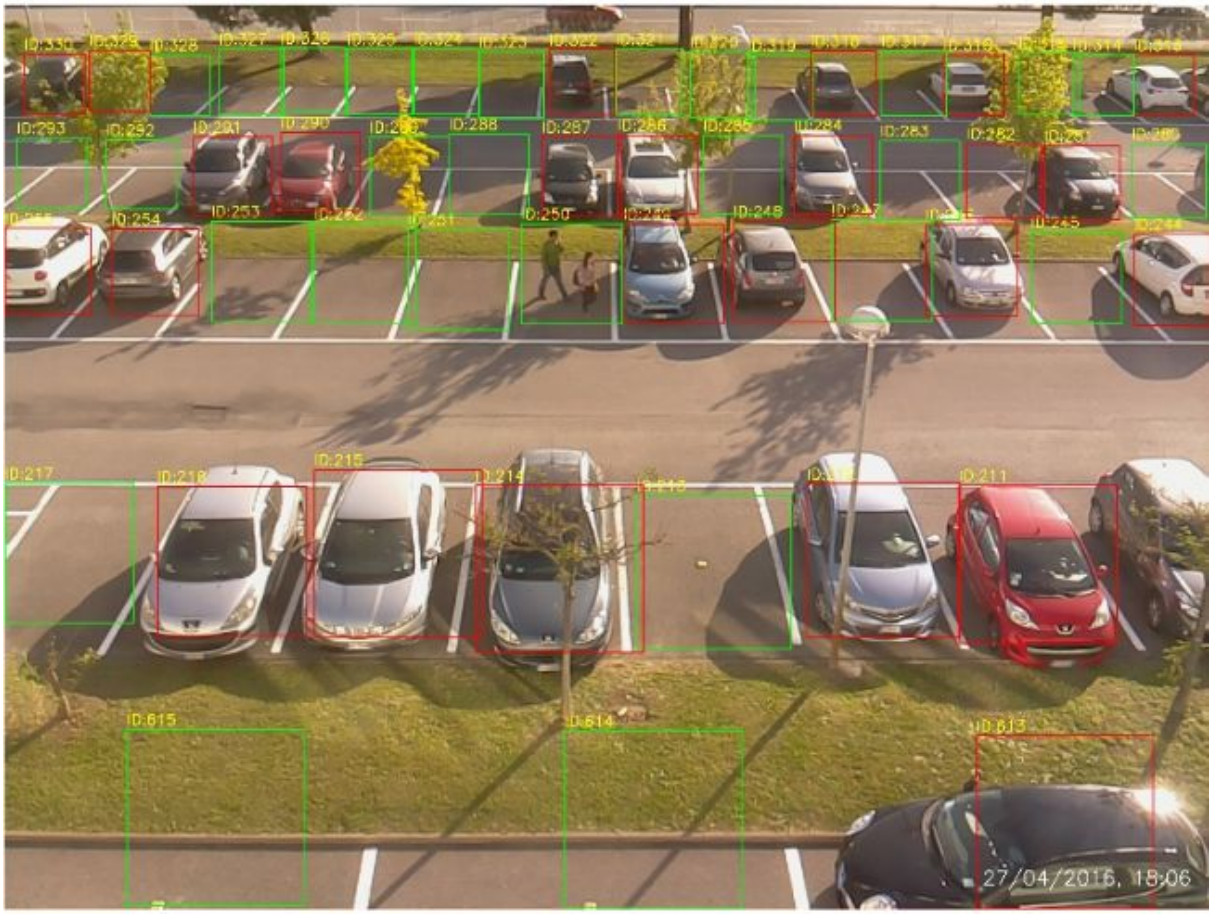
Free Slot

After training, we test our model on the validation set. So far we are able to tell if an image contains a vehicle or it's free. But we need to implement this for the complete image.

Applying the model to the whole system:
To do that I used the CSV file which I mentioned previously. Since the position of the initial point, width and height are given I can crop using `cv2.rectangle.` After the cropping is done necessary preprocessing is done on the image and is given to our model to predict the class. Based on the type of class I will color that particular rectangle with Green or Red color.
If the slot is empty I will cover the slot with a green rectangle and if the slot is full I will cover the slot with a Red rectangle and to do that the following code is used

```
if result[lot_id] ==0:
        img = cv2.imread('input.jpg',cv2.IMREAD_COLOR)
        cv2.rectangle(img,(x1,y1),(x2,y2),(0,0,255),1)
        cv2.imwrite('input.jpg', img)
    else:
        img = cv2.imread('input.jpg',cv2.IMREAD_COLOR)
        cv2.rectangle(img,(x1,y1),(x2,y2),(0,255,0),1)
        cv2.imwrite('input.jpg', img)
```

After performing this step for all the patches the resulting images will be as follows



Since I am unable to train this model on my own I am adding the output image of the pertained model. As you can see the slots with the presence of a car are painted with Red rectangle and the empty slots are painted with green slots.

Resource:

- [1] Deep learning for decentralized parking lot occupancy detection
    - G Amato, F Carrara, F Falchi, C Gennaro, C Meghini, C Vairo
- Online platforms like Medium and Towards Data Science

As I mentioned I Implemented this code but I didn't train it and here is the link for Source code

GitHub link: **https://github.com/srinivasmachiraju/Car-Parking-Guide**

Further improvements:

This model can be implemented further to a live streaming video which is a collection of sets of images and doing this makes our model work in real-time.

Conclusion:

I hereby conclude my report and I would like to thank JIO for giving such a real-world problem and helping me to gain intuition on how to apply theoretical knowledge to the practical world. I would like to thank Techgig as well for hosting such a beautiful contest.

Thank you
Srinivas Machiraju
Team CG_TEAM_IGZYD