Parking Space Classification using Convoluional Neural Networks

Jordan Cazamias & Martina Marek
Stanford University
jaycaz@stanford.edu & mmarek@cs.stanford.edu

Abstract

Our overall goal is to develop an automated system to count the empty parking spaces in a parking lot, given only an image of the lot (and, optionally, the bounding boxes of the spaces) as input. Two possible approaches are explored to achieve this: a binary classifier that can label an individual space image as empty or occupied, and a multinomial classifier (a.k.a. counter net) that returns the number of empty spaces given the entire lot image.

The binary classifier has yielded very encouraging results, with over a 99% overall accuracy and strong indications that it would generalize well to other parking lots. However, the major downside is that it requires the bounding boxes of the parking spaces, limiting its usefulness to stationary camera feeds. The counter net has an overall accuracy of about 81%, much lower than the binary classifier, and its errors exhibit a wide variance. This is a useful baseline, but more improvements should be made to make it more practical for real-world use. If this approach can be improved, however, it would be the more useful of the two since it only requires the image as input.

1. Introduction

Finding a parking spot, especially in a city or during busy hours, can be a tedious process. According to a study by Donald Shoup at UCLA, people in Westwood Village (with a population around 50,000) drive over 30 extra kilometers each year to find vacant parking spaces (Shoup 2006). On average, people spend 8 minutes finding a spot (Shoup 2008). It would be not only more convenient, but also environmentally friendlier, if drivers had up-to-date information on where the nearest empty parking spots are and could plan accordingly. Another possible use case for such a system is the home security sector. Especially in poorer countries, theft is a big problem and user might want to detect whether their car is in the drive-way or not, and be notified in case it was moved to prevent theft.

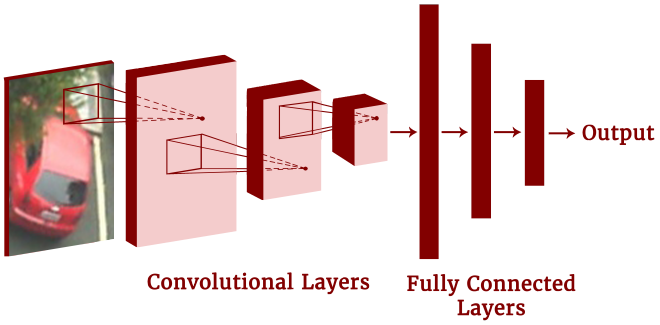

Figure 1. Diagram of our CNN architecture for both the binary classifier and the counter net

There are two major approaches for the detection of empty parking spots. Sensors could be pre-installed in every parking spot to detect the presence of a car. Alternatively, visual information (typically collected from pre-install security cameras) could be used to determine if a parking lot is empty or not. The first approach is quite expensive to realize, while the second approach could leverage existing camera systems and therefore be much more feasible.

We will be working on a parking lot occupancy classifier, and will create two systems to attempt this task. Our first system is bottom-up, where the parking lot image is pre-segmented into individual parking space images, and a classifier decides whether each space image represents an empty or occupied space. We use a Convolutional Neural Network (CNN) as the classifier. As this is a binary classification, this should prove to be a fairly simple task for a CNN. Despite the pre-segmentation restriction, such a system could still be practical as it could be applied to any stationary camera feed where the regions of interest (i.e. the bounding boxes of each parking space) remain fixed. One could use pre-existing cameras (like security cameras) for this task, or install one or more cameras that overlook

the whole parking lot.

Our second system is top-down, where we feed images of the entire parking lot to the CNN without segments and let it output the number of free parking spaces. We treated this as a classification problem, where the classes are the number of empty spots, which are capped at some maximum value n. This works fine for a real-world application since after, say, 20 spaces are empty, it does not matter how many more parking spots are empty. This is a much harder problem, since there are much more than two classes and because the CNN has to count the absence of objects (i.e. cars). Furthermore, the location of the empty spaces is not exactly known, so additional localization would need to be done to pinpoint the spaces' locations. The upside of this approach is that the parking lot images do not need to be pre-segmented, and thus there is no requirement that the images be from a stationary camera feed. This opens up the possibility of using alternative image sources, such as drones or even satellites.

2. Related Work

There are some papers that tackled the parking space occupancy problem, but so far nobody seems to have used CNNs on that kind of problem, although they are the state-of-the-art since the breakthrough of AlexNet (Krizhevsky, Sutskever, and Hinton 2012) in 2011. Most approaches we found used linear classifiers, like SVMs or KNN, to classify empty parking spaces. We hope to improve their results by using a CNN.

Nicholas True describes an approach for classification of vacant parking spots that used both SVMs and KNNs with color histogram and corner detection features for classification, which achieved around 90% accuracy in "Vacant Parking Space Detection in Static Images". The results reported by Fusek et al., "AdaBoost for Parking Lot Occupation Detection" who used an Adaboost based algorithm are in the same range.

Wu and Zhang, "Parking Lots Space Detection" describe another approach that uses a multiclass SVM trained on a Gaussian color model. The best results obtained are around 84%.

For a CS229 class project, "Parking Spot Detection from Aerial Images" detected parking spots on aerial images. They used a variety of different features that captured geometrical, statistical and optical information and trained a linear SVM on those. With this approach, they achieved accuracies over 90% on their dataset.

An example that solves the more general problem of counting objects in a scene is Zhang et al. 2015, which uses a CNN-based framework to estimate the number of individuals in an image of a crowd. We will use similar error metrics to analyze our counter net's performance.

One paper that is also worth mentioning is that provided with PKLot, our dataset of choice ("PKLot - A Robust Dataset for Parking Lot Classication"). Besides explaining the details of the dataset, it also provides classification baselines and proposed research directions, which were an important influence on the metrics we used to test the binary classifier.

Figure 2. Segmented parking lot

3. Methods

3.1. Convolutional Neural Nets

The workhorse of our algorithm is a Convolutional Neural Network, a deep learning data structure that works particularly well for computer vision tasks. A CNN consists of multiple convolutional layers. These layers scan over the image, using a small, $k \times k$-pixel feature detector called a filter, and mark the regions of the image that align most closely with that filter. It does this by sliding a $k \times k$ window over the image and computing the dot product between that window's pixel values and the filter's pixel values. Unlike other vision algorithms that use a fixed set of features, such as edge detectors, a CNN learns its own features via machine learning, making them rather well suited for the data at hand. For instance, a CNN trained to detect faces may learn to look for shapes such as eyes and noses, while a CNN trained to classify advertisements may learn to look for company logos.

An entire CNN typically has multiple convolutional layers, each of which also uses multiple filters. The conv layer at the bottom of the net receives the raw image data, and uses each of its filters to identify the features that it deems important. Each filter produces a 2-dimensional array of dot products, called an activation map. The activation maps for all the filters in one layer are sandwiched together into

a 3-dimensional activation volume. This volume is the output of the conv layer. After leaving the conv layer (and going through an activation layer such as ReLU or tanh), the data then typically goes through a max pool layer that scales down the activation volume, usually by a factor of 2. This downsampled activation volume then becomes the input for the next conv layer.

This cycle (Conv layer, then ReLU layer, then Maxpool layer) happens several times, depending on the size of your CNN. The results from the last conv layer are fed into a vanilla neural network with one or more fully connected layers. Finally, the output of this last fully connected layer goes through a softmax layer which converts the fully connected layer's output into a set of class probabilities, one for each class. It does so using the following formula. Given the correct class label i, the input vector $\vec{x}$, which is fed through the network, and renders the scores $\vec{s}$, the output for the softmax is the following:

$$h_\theta(x) = \frac{\exp(s_i)}{\sum_{j=1}^{k} \exp(s_j)}$$

To improve the CNN's performance, a loss function is used to estimate the quality of the CNN's answer (i.e. its class scores). We used the negative log likelihood as the loss function, which is the following:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}(1-y)\log(1-h_\theta(x)) + y\log h_\theta(x)\right]$$

Minimizing this function, then, will allow the CNN to maximize the probability that it chooses the correct output given the input.

3.2. Batch Normalization

The performance of a CNN often depends heavily on the initialization of the parameters of the network, and since those are initialized randomly, the performance can vary widely even when the same hyperparameters are used.
To solve this problem, one can either train the model multiple times with the same hyperparameters to get the best possible performance, or normalize the input to each layer with a technique called batch normalization (Ioffe and Szegedy 2015). To describe batch normalization in more detail:

During training, each dimension of the mini-batches is normalized using the mean and variance of that specific batch. However, to keep the expressiveness of the model, additional parameters β and γ are introduced, that scale and shift the normalized value and are learned during training. (Just normalizing, without giving the net the option to scale and shift the normalization, would constrain the net to specific outputs and therefore reduce its expressiveness).

The algorithm then looks as follows, as described in Ioffe and Szegedy 2015:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 3. Batch normalization algorithm

Batch normalization is usually added to the network after convolutional or fully-connected layers, but before the non-linearity. During test time, the normalization is done using a sample mean and variance.

During the training process for the binary classifier, we ran into the above issue. Occasionally, due to an unlucky parameter initialization, the validation accuracy of our model would plummet. After applying batch normalization layers after each convolutional layer, this problem disappeared.

3.3. Adam Update

After the forward and backward pass of a CNN, the parameters of the net need to be updated using the gradient found during the backward pass. The simple way to do this is with Stochastic Gradient Descent (SGD), which adds a scaled amount of the gradient to the new parameters:

$$\theta_{i+1} := \theta_i - \alpha\nabla J(\theta_i)$$

However, alternative update methods have been proposed to speed up the learning process. The update rule we used almost exclusively is adam, the full algorithm for which can be found in Kingma and Ba 2014. For our models, there did not seem to be any marked difference in the two update rules other than the speed. Adam update was much faster, allowing our models to achieve the same training loss in fewer training epochs.

3.4. Saliency Maps

In order to analyze our trained model, we implemented Saliency Maps, as described in Simonyan, Vedaldi, and Zisserman 2013, to be able to see which regions in the image our trained net pays most attention to given a specific class. By back-propagating the derivative of the class score through the net, one can learn which spatial regions in the image contributed most heavily to that score.
Or, more formally, one computes the forward pass on a given image (in order to compute the cache needed for the backward pass). Then, instead of computing the loss, the derivative of the class l_j we want to compute the Saliency map for is set to 1, while all other derivatives are 0 (this vector $dout$ has the same size as the output of the net). The vector $dout$ is then back-propagated through the net and outputs dx, the derivative on the image:

$$dx = \text{Backward(dout)},$$

$$\text{where } dout_i = \mathbb{1}[i = j], \text{ and } |dout| = |l|$$

To generate a single class saliency for each pixel, the maximum of the absolute values over all channels is taken for each pixel Simonyan, Vedaldi, and Zisserman 2013:

$$M_{ij} = max_c |dx_{ijc}|$$

4. Dataset

The dataset we used for this problem is PKLot "PKLot - A Robust Dataset for Parking Lot Classication" (`http://web.inf.ufpr.br/vri/news/parking-lot-database`). It provides 12,000 images taken from three different camera feeds of two different parking lots. The images were taken over a 30 day period at 5 minute intervals. Each parking lot image is annotated with the date, time and current weather conditions (either sunny, cloudy, or rainy).

The images are further segmented into over 600,000 sub-images of individual parking spaces. Each space is hand-labeled as vacant or occupied. The individual spot images have also been transformed to remove the perspective distortion and are all rotated to a vertical orientation (although not all the images are the same size). Examples can be seen in figures 2 and 4.

5. Implementation

To reiterate, we took two approaches to solving the problem of counting the empty spaces in a parking lot image. The first is a bottom-up binary classification system, where we feed each individual parking space image to a CNN and it chooses whether that space is vacant or occupied. The second approach, which we will call the counter net, is a multinomial classifier that receives an image of the entire parking lot and returns a count for the number of empty spaces in that image, up to a predetermined max value n. Training images with more than n empty spaces were reported to the CNN as having n empty spaces.

Figure 4. Examples of occupied and empty spots in the dataset

5.1. Architecture of the CNN

Figure 1 illustrates the architecture of our CNN for both the binary classifier and the counter net. The details for both follow.

5.1.1 Binary classification

For the binary classification model, we used a fairly simple architecture with three convolutional layers, each with a depth of 10, 20, and 30, respectively, and a receptive field of 5x5. Each of those layers was then followed by a batch normalization layer and a ReLu activation function. After that, a max pooling layer with a filter size of 2x2 was added. The output of the convolutional layers was then fed into 3 fully-connected layers with 30, 20, and 10 neurons, respectively. Here we did not use batch normalization; each of the layers was directly fed into a ReLu activation. Finally, we used a Softmax layer to compute class scores.

5.1.2 Counter Net

The counter net's architecture was identical to the binary classifier's, with the exception of the input size and the class labels. Because the counter net uses images of the entire parking lot, a much larger input size was required. As for the class labels, there were now $n + 1$ labels where n is the predetermined maximum count.

5.2. Preprocessing

For the binary classification, we used the pre-segmented images of parking spaces, which were already rotated

Model	Accuracy
SVM with Gaussian Color Model (Zu et al.)	83.6%
kNN with Color Histogram (True)	89.0%
SVM with Color Histogram (True)	94.0%
Adaboost (Fusek et al.)	95.5%
CNN	**99.97%**

Table 1. Comparison of the accuracy of the CNN compared with the classifiers from section 2

to a vertical orientation. The images are different sizes, however, so they were all resized to 48x64 before training. Furthermore, all the images were normalized so the mean of each color channel was 0.

For the counter net, the images were resized to 256x128 pixels and normalized in the same way as the pre-segmented parking spots.

The data and associated metadata was also prepackaged into a single HDF5 file so it could be efficiently piped to our model in batches (The HDF Group 1997-).

5.3. Technical Approach

We used the *Torch* (`http://torch.ch/`) framework to build and train our CNNs and DeepMind's torch-hdf5 package to process the compressed HDF5 version of our dataset. For various other scripts, such as compressing the dataset, running cross validation, etc., we typically used Python with NumPy.

6. Results

6.1. Measurement for performance

6.2. Binary classification

To measure the performance of the binary classifier, we used accuracy, the number of correctly classified examples divided by the number of total examples. As expected, the CNN performed very well on the classification task and scored an accuracy of **0.9997** on the test set, even using a simple architecture with only 3 convolutional and fully-connected layers with a fairly small amount of neurons. Using a CNN on this task therefore outperformed all previous approaches considerably, as described in section 2, that used feature-based classifiers. A comparison of the performance of different approaches can be seen in table 1.
Using a simple CNN for this task is therefore a successful approach and has the advantage that no features need to be defined. It is also robust to overfitting, scoring over 99% for the training, validation and test set, and converges within one training epoch.

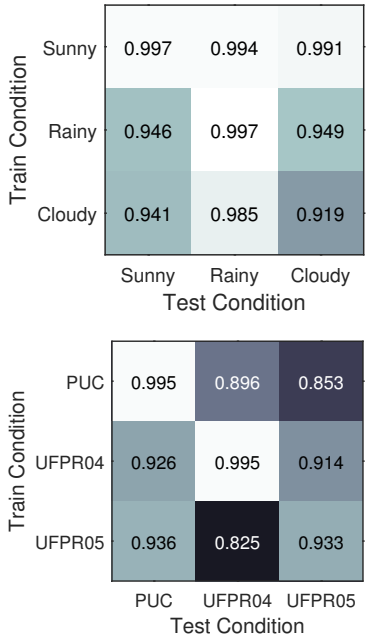

Figure 5. Confusion matrices for our binary classifier based on weather conditions and parking lot camera feeds, respectively. For each result, the CNN was trained on the condition of row i, then tested on the condition of column j.

6.3. Generalization

To test how well our model generalizes across different parking lots or weather conditions, we trained our net on a specific condition and tested how well it performed on the other conditions.
During training, we used the hyperparameters we found earlier for the binary classification and performed a 5-fold cross validation to make sure the performance is not influenced by different initializations. We used 70% of the images of a specific condition for training and 20% for the validation set, and tested on the remaining 10% of this condition. While testing on the other conditions, we again used 20% of the data for the validation set, and 10% for the test set.
More specifically, we trained it on each of the three weather conditions in our training set, namely *sunny, rainy, and cloudy*, and then tested the test performance on all other weather conditions for each of those, rendering the 3x3 confusion matrix in figure 5.

It can be seen that the performance on the test set varies as the weather conditions are changed, hinting that when the net is trained on the entire dataset, it learns to adapt to different lighting conditions. However, even when trained on one lighting condition, the net stays above 90% in all cases, which is fairly robust. The best performance over all weather conditions was obtained when training on sunny

weather conditions.

In a second step, we tested how well the net generalizes over images of different parking lots. Here, the main differentiator between the images is the camera angle, resulting in a different perspective distortion. The images in our dataset are drawn from two different parking spots (namely, PUC and UFPR), where the UFPR lot had two different camera feeds (UFPR04 and UFPR05). Again, we trained on one of the camera feeds and tested the CNN on the other ones. The results can be seen in 5. The CNN also performed reasonably well, though not as well as during the weather test. This suggests that the CNN is more sensitive to changes in the viewing angle of the parking spaces than it is to changes in lighting.

6.4. Counter Net Error

The overall test accuracy for counter net was **0.8084**. However, unlike the binary classifier, not much can be said about the counter net's performance by just looking at the accuracy. Therefore, we will perform a deeper dive into the counter net's performance.

Because the problem of counting parking spaces is a quasi-regression problem, we should not only consider how often the counter net correctly chooses the exact count, but also what it chooses when it is wrong and how far that choice deviates from the true count. After training on the entire dataset of parking lot images, Figure 6 gives us a matrix displaying the model's answer versus the correct answer on the test data.

Ideally, we would like our model to learn some kind of locality, i.e. even when it makes the wrong choice, its choice should be as close to the correct choice as possible. If that were true here, this matrix would exhibit a clustering of values along the major diagonal line. Unfortunately, that does not seem to be the case. While there may be some promising clustering in the middle, the various vertical bars indicate that the model still likes to gravitate towards some values regardless of what the ground truth label actually is.

One encouraging note about the matrix, however, should be explained. The vast majority of the test data - about 1,000 out of the 1,200 tested - was in the small box on the bottom right, which counts when the model guessed there were "19 or more" empty spaces and was correct. In essence, the counter net is still quite accurate at noting when a parking lot has a large number of empty spaces and when it is almost full.

We can go further into the analysis of the counter net error by examining the distance between its choices and the correct answer. Figure 7 illustrates such a distribution.

Furthermore, we can extract some useful statistical data from this distribution:

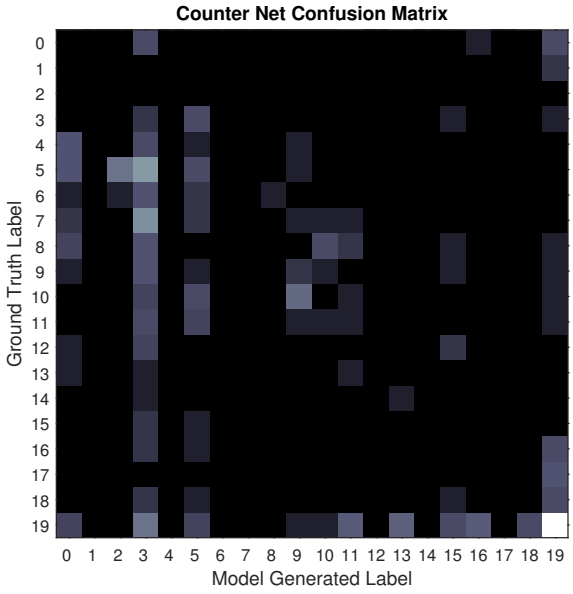

Figure 6. Comparison between the ground truth empty space counts and the counts found by the CNN. A lighter box indicates that the (ground truth, model choice) pair occurred more often.

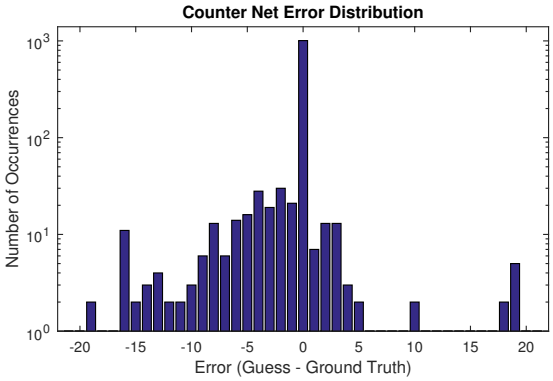

Figure 7. Distribution of errors between the ground truth empty space counts and the counts found by the CNN. Negative values indicate the model underestimating the number of empty spaces in an image, and vice versa for positive values.

Mean Error	-0.5805
Std. Dev	3.1824
Sample Size	1242
95% Confidence Interval	[-0.7575, -0.4035]

We will also consider the Mean Absolute Error (MAE) and Mean Squared Error (MSE), which were used as error metrics in a similar CNN counting system (Zhang et al. 2015):

Mean Absolute Error (MAE)	1.0749
Mean Squared Error (MSE)	10.4565

Judging by the mean and confidence interval, our counter net is biased towards a negative error, i.e. it tends to underestimate the number of empty spaces in an image. While it may be better in practice for such a system to consistently underestimate rather than overestimate, we would ideally like to drive this mean closer to 0.

We would also like to see the MAE and MSE brought down overall. Currently there is no clear basis to say what would be "good" values for the MAE and MSE, but they can at least be used to compare different versions of the counter net. Therefore, the current values will serve as a useful baseline when improving the counter net in future work.

6.5. Saliency maps

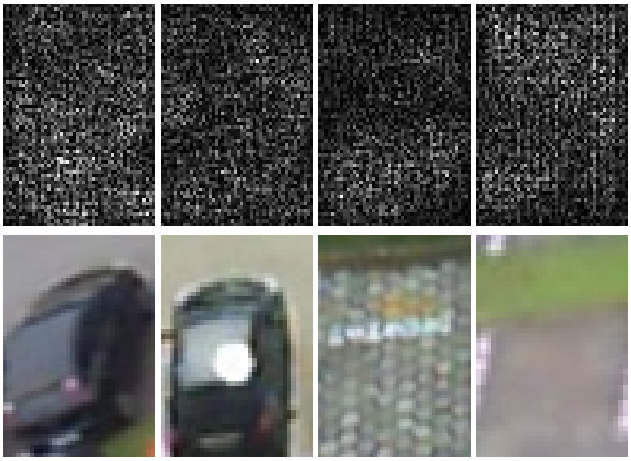

Figure 8. Example saliency maps for binary classification

To analyze which regions of the image the CNN pays attention to when classifying an image, we implemented saliency maps as described in Simonyan, Vedaldi, and Zisserman 2013.

6.5.1 Binary classification

During the binary classification, the CNN seems to spread its attention all over the image, without any noticeable attention on key features of any of the two classes. We thought that it might pay attention to straight white lines for empty parking spots, or focus the attention on the middle part of the image for occupied lots, but it seems like it considers all regions of the image as important for the classification. Results can be seen in figure 8.

6.5.2 Counter Net

For the classification of the number of empty parking spots over the whole image, the net seems to concentrate its attention on specific areas of the image, but unfortunately it does not seem to only concentrate on the empty lots, but also on the occupied lots. Using the saliency maps for localization is therefore not possible. Examples of the picture of the whole lots and their saliency maps can be seen in figure 9.

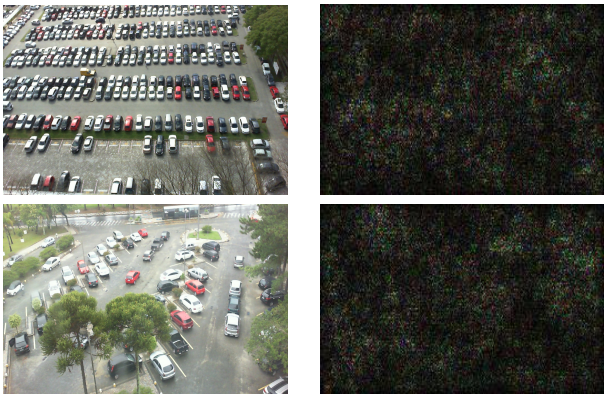

Figure 9. Example saliency maps for counter net

7. Conclusion

The binary CNN clearly outperformed the classifiers described in other papers in the classification task and rendered really good and stable results.
For the counter net, the results are fairly good when considering that it is a much harder problem. For a reliable system, however more improvements would need to take place. Firstly, although we performed cross validation with different architectures and hyperparameters, one could try even more complex architectures and a wider range of hyperparameters. Secondly, while we treated it as a classification task, one could also define it as a regression task (by using a regression loss instead of Softmax loss), and see if the performance improves. This is a very simple adjustment to the net architecture, but unfortunately we had no more time to run this test on the GPU. Additionally, if the number of total parking spots is known, one could also count the number of cars in the parking lot and subtract it from the number of total spots. This is probably an easier problem, since the CNN then has to count the number of objects of a specific kind in the image, instead of the absence of objects.

7.1. Future work

For the future, there are many ways on how to improve these models. Some of the ways we would like to expand this topic are outlined below.

Counter Net Regression

First, the counter net could be significantly improved if it were treating the space counting as a regression problem instead of multinomial classification. Thus, rather than using softmax as our criterion and looking at Mean Squared

Error for our results, it would likely be worthwhile just to train the counter net on MSE directly, or on some other regression loss. That way, it would be more likely to learn the locality of errors that this type of task possesses, and would ideally result in a more predictable and less varied error.

Spatial Transformer Modules

Second, it would be important to make both models rotation-invariant in order to be used on a wide variety of parking lots. While the segmented parking spaces on our lots cover different angles and our classifier therefore learns how to cope with spaces in different rotations, the rotations covered are limited and having a model that is completely invariant to rotations would be useful. For this, one could implement Spatial Transformer Layers, a module which can learn how to perform an affine transform on the feature map and therefore becomes invariant to rotations. More details on Spatial Transformer Layers are described in Jaderberg et al. 2015.

Satellite Imagery

Given more time and resources, one could also collect another training set on satellite images, to test how well the network generalizes to images taken from a radically different angle, namely the extreme perspective of a top-down satellite view. Additionally, this would show if satellite images could be used to implement such a system for a user application.

Reduced Positional Data & Attention Models

It would be ideal to eliminate the need for bounding boxes around the parking spaces. However, rather than jump from requiring 4 coordinates (the corners of the bounding box) per space to 0 coordinates per space, it would also be worthwhile to consider intermediate constraints. For instance, could a system be devised that only required one coordinate, such as the central coordinate of the parking space? Given this one point on an image of an entire parking lot, all the CNN would need to do is determine, roughly, which pixels around that point correspond to that space. One possible approach would be to use a guided attention model (as touched on in Tang, Srivastava, and Salakhutdinov 2014) to pick out the regions where the parking spaces are and feed these regions into a binary classification CNN.

Parking Space Detection

Finally, one could, in theory, automatically find the bounding boxes for the parking spaces using detection. While detection systems exist, the major challenge for this problem would be that most state-of-the-art detection systems do not support rotated bounding boxes. Therefore, an implementation of the Spatial Transformer Module (Jaderberg et al. 2015) would be useful to make our models rotation invariant. Then unrotated bounding boxes could be used for the detection problem. This would likely be the most challenging way to improve this system, but if it were to work well in practice, it would open up the applicability up to any live camera feed.

References

Almeida, P. et al. “PKLot - A Robust Dataset for Parking Lot Classication”. In:

Fusek, Radovan et al. “AdaBoost for Parking Lot Occupation Detection”. In:

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167. URL: http://arxiv.org/abs/1502.03167.

Jaderberg, Max et al. (2015). “Spatial Transformer Networks”. In: *CoRR* abs/1506.02025. URL: http://arxiv.org/abs/1506.02025.

Kabak, Mehmet Ozan and Ozhan Turgut. “Parking Spot Detection from Aerial Images”. In:

Kingma, Diederik and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Shoup, Donald (2008). “City of Dreams”. In: *Tablet* 18 March.

Shoup, Donald C (2006). “Cruising for parking”. In: *Transport Policy* 13.6, pp. 479–486.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2013). “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034. URL: http://arxiv.org/abs/1312.6034.

Tang, Yichuan, Nitish Srivastava, and Ruslan R Salakhutdinov (2014). “Learning generative models with visual attention”. In: *Advances in Neural Information Processing Systems*, pp. 1808–1816.

The HDF Group (1997-). *Hierarchical Data Format, version 5*. http://www.hdfgroup.org/HDF5/.

True, Nicholas. “Vacant Parking Space Detection in Static Images”. In:

Wu, Qi and Yu Zhang. “Parking Lots Space Detection”. In:

Zhang, Cong et al. (2015). “Cross-scene crowd counting via deep convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 833–841.