Parking Spot Occupancy Detection on the CNRPark+EXT Dataset Using Assorted Deep Convolutional Neural Networks

Anthony Harris, Joshua Ellis, Naseem Saquer

Missouri State University, Springfield, Missouri

***Abstract*— Automated parking space occupancy detection can be a challenging, yet highly rewarding task in areas of high population density and limited parking availability. Weather, time of day, visual occlusion, and shadows all add to the complexity of the task. To that end, we use the CNRPark+EXT dataset to train several diverse Convolutional Neural Networks to be able to classify a given parking space as either "Busy" or "Free". We compare the results of very large and very small models with parameter counts ranging from 147 million all the way down to 6 thousand. We show that smaller models can achieve comparable levels of accuracy as their larger counterparts with a fraction of the number of parameters.**

Keywords—Image Classification, Deep Learning, Convolutional Neural Networks, Residual Neural Networks, Parking Space Occupancy Classification

I. Introduction

It is predicted that by 2050 6.3 billion people (66% of the global population) will live in cities; this number is considerably larger than the 3.9 billion (54% of the global population) that currently live in them [1, 2]. Given the already challenging situation with respect to finding a parking space in densely populated cities like New York and Los Angeles, it is expected that this issue will only become more prevalent as more and more motor vehicles are introduced into these environments. As such, the need for an intelligent, automated means of conveying the availability of parking in a parking lot to drivers in need of parking is higher than ever. This is further accentuated by the recursive nature of traffic congestion; when a driver fails to find a parking space, they then re-enter the flow of traffic, further adding to the congestion. This additional congestion then creates more search time for drivers, who then re-enter the flow, so on and so forth. Studies have found that in densely populated cities across the world, approximately 30% of inner-city traffic is searching for an available parking space at any given point in time [1].

There are several potential approaches to this problem; the naïve approach is to simply have a human monitor the occupancy status of a parking lot and update this information on some publicly available resource that can be accessed remotely. This is neither cost effective, nor viable; the resources simply are not there to be able to pay someone to manually monitor every parking lot in a densely populated area. A somewhat less naïve approach is to use a lot-scale counter system that tracks the number of available parking spaces based upon the number of vehicles that cross over some threshold into and out of a parking lot; the number of available spaces would then be presented on a billboard or some other means of information conveyance [3]. This method is limited in that it considers the parking lot as a whole, as opposed to each individual parking space. Parking lots can become very large, very quickly; just knowing that a parking space is available somewhere in a lot may not always be sufficient. We argue that it is more desirable to know *which* parking spaces are available so that a driver may easily navigate to an available space instead of searching through a lot. This can be made possible by classifying the occupancy status of each parking space in a parking lot and then conveying the information of the parking lot to searching drivers; in order to accomplish this goal, we must first determine a means of accurately and efficiently classifying the occupancy status of each parking space in a parking lot.

II. Background

We hypothesize that one of the easiest, most efficient ways of performing automated parking space occupancy detection in a parking lot is by using a Convolutional Neural Network (CNN) to determine the status of a parking space given only an image of said parking space. As is the case with many Deep Learning models, CNNs require a large amount of data to train on. To that end, we turned to the CNRPark+EXT dataset provided by [4]; this dataset consists of approximately 150 thousand images of parking spaces that are classified as either "Free" or "Busy". These images are further categorized by the weather at the time of capture; the weather types are Sunny, Overcast, and Rainy. Minimal preprocessing was needed to use these images; most of them were 150x150 when accessed, and the rest were resized to match. The distribution of classification, weather, and class by weather can be seen in Tables 1, 2, and 3 respectively.

Figure 1: Sample images from CNR-EXT [4, 6, 11]

Class Frequencies

Class	Count
Free	65658
Busy	79307

Table 1

Weather Frequencies

Weather Type	Count
Sunny	63178
Overcast	44243
Rainy	37544

Table 2

Class Frequencies by Weather

Class	Weather Type	Count
Free	Sunny	25665
	Overcast	21067
	Rainy	18926
Busy	Sunny	37513
	Overcast	23176
	Rainy	18618

Table 3

A major challenge in being able to automatically classify the occupancy status of individual parking spaces lies in compute; deep CNNs, while highly performant on image classification, also tend to be computationally intensive due to their sheer size. We define the size of a model to be the number of parameters therein; as such, a model with fewer parameters is considered smaller than a model with more parameters, regardless of the width or depth of the models in question.

To solve the compute problem while obtaining the highest possible accuracy, we focus on the use of various model architectures, with parameter counts ranging from 147 million down to a meager 6 thousand, and we report on the accuracy achieved by each model. We use models that are predefined and well explored, as well as some models of our own design.

We used TensorFlow 2.3.1 and Keras 2.4.3 with Python 3.8.5 to define and run all models used in our experiments. The code for this project is available at https://github.com/KillerBOB999/CSC790_Project.git and is complete with visualizations for each of the models we implemented.

III. Methods

A. AlexNet / Mini AlexNet

One of the first network architectures that we applied to the CNRPark+EXT dataset was the AlexNet, which is one of the most popular CNN architectures to date and was first proposed in [5]. The AlexNet is a very deep model with 51 million parameters. Despite its linear, relatively simple makeup seen in Figure 2, the AlexNet architecture is notable for its performance on the ImageNet classification problem where it beat the state-of-the-art model's top-5 error rate by 11% in 2012. Each layer in the AlexNet utilizes the Rectified Linear Unit (ReLU) activation function for non-linearity. The usage of ReLU for such large networks is notable since it was found in [5] that CNNs could be trained much faster than when using saturating activations like sigmoid or hyperbolic tangent (tanh). In fact, equivalent networks using tanh as the activation function have been shown to converge roughly six times slower; faster learning greatly helps the accuracy of large models training on large datasets. There are 5 convolutional layers in the model, with each layer being immediately followed by batch-normalization and max-pooling. Each of the 3 fully connected layers utilizes a 40% dropout in addition to the batch-normalization mentioned before. These techniques work in conjunction to prevent such a large number of parameters from overfitting on the training dataset. Our implementation of the AlexNet differs from that used in [5] in that the size of our input images remained at 150x150 instead of the 224x224 that is traditionally used. Additionally, since this network was being applied to a binary classification problem, only one output neuron was used.

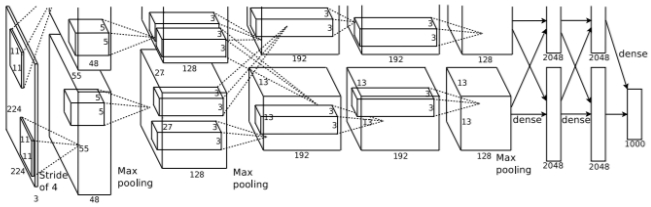

Figure 2: Visualization of the AlexNet [5]

In terms of performance, the AlexNet achieved an accuracy of 98.59%. Relative to other architectures explained later in the paper, this performance is on the better side of things, but not by a wide margin. It is worth noting, however, that two networks with less than half the number of parameters of the AlexNet achieve higher accuracies on the testing set. With this result in mind, we implemented the mini AlexNet model (mAlexNet) defined by [6, 11] to compare the performance. The mAlexNet, which is heavily inspired from the AlexNet in [5], has 20,000 training parameters, making it take up less than 1% of the size of the full AlexNet. It was found that due to the simplicity of the parking space classification problem in comparison to the ImageNet problem, the mAlexNet performed well on the CNRPark+Ext dataset despite the large reduction in size. In fact, the mAlexNet trades off 0.62% accuracy on the testing set in exchange for 24 million fewer training parameters.

B. SimpleResNet

Given the success of the AlexNet and mAlexNet, we conjectured as to how a more modern architecture like a ResNet [7] would perform. ResNets are unique in that they counter the exploding and vanishing gradient problems frequently encountered in deeper neural networks; they do so by designing residual connections where the output of a layer is added to the input of a layer some number of layers farther down the network architecture. Figure 3 shows an example of what this residual connection looks like graphically. While there are several existing ResNet model architectures provided by Keras and Tensorflow [8], we chose to build a small, simple ResNet model based upon the example given in the Tensorflow documentation at [9]. Our model comprised of only 2 of these residual connections, preceded by 2 convolutional layers and a single max pooling layer. Following the residual connections were a single convolutional layer, a single global average pooling layer, 3 dense layers, each with 256 nodes, and a single dropout layer to prevent overfitting. This model had a total of 352,770 parameters, giving it considerably more than the Mini AlexNet, but much less than the full AlexNet. We found that this model was able to achieve an accuracy of 98.15%.

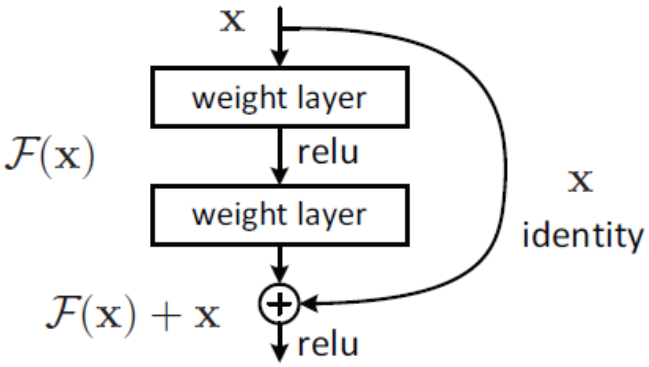

Figure 3: Example of a Residual Connection [7]

C. DenseNet / MiniDenseNet

Like the ResNet [7], the DenseNet model architecture [10] aims to counter the exploding and vanishing gradient problems by defining residual connections between layers. The DenseNet architecture, however, connects every layer to the following set of layers in each Dense Block via concatenation. Figure 4 demonstrates what a Dense Block might look like; it shows that a block is comprised of some number of groups of batch normalization and convolutional layers, all of which feed their outputs into all following groups in the block. This allows the information from each layer to be propagated farther in the model, which promotes high feature propagation [10].

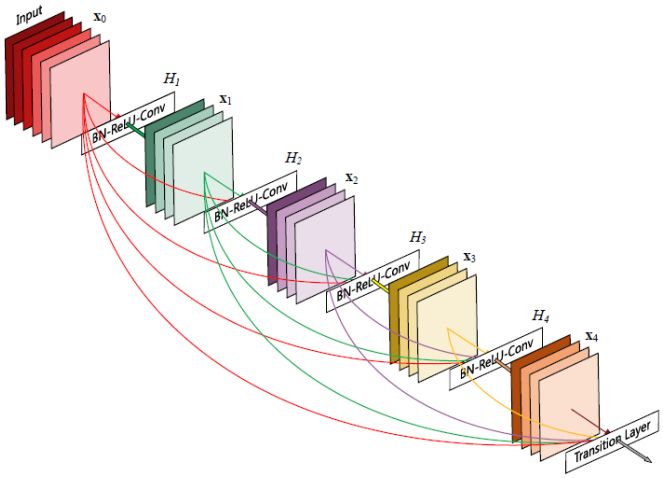

Figure 4: Example of a Dense Block [10]

Our simple DenseNet implementation contained 6 of these groupings in a single Dense Block, preceded by only a convolutional and max pooling layer. The Dense Block is followed by a single global average pooling layer. This architecture resulted in a total of 361,794 parameters; 358,914 of which are trainable. This is notably comparable to the simple ResNet implementation, and performed very similarly, achieving an accuracy of 97.97%.

To further improve on the small parameter count and high accuracy achieved by the mini AlexNet from [6, 11], we designed and implemented a comparable model using the DenseNet architecture as a baseline. This was accomplished by modifying the hyperparameters used to define our DenseNet model; we reduced the number of groups in the Dense Block to only 2 and reduced the number of filters in the convolutional layers down from 32 or 64 to only 16. These small changes resulted in a drastic decrease in parameters; our mini DenseNet (mDenseNet) only contains a total of 11,026 parameters, 10,866 of which are trainable. This is approximately 51% of the size of the mAlexNet, and it achieved an accuracy of 97.43%.

D. SimpleNet

We implemented SimpleNet which takes the same 150x150 images as the other networks but SimpleNet uses average pooling with 50x50 pools to reduce the images to 3x3. SimpleNet then uses a multi-layer perceptron with two 64 node hidden layers to classify the dataset. This model was implemented as an experiment to understand what the other models might be learning and to quantify the baseline accuracy one might expect from classifying the CNRPark+EXT dataset. SimpleNet is intended to be a look at how well a network could perform if all detail was removed from the images and only the rough color gradient of the image remained. With only 6,082 parameters, it achieved an accuracy of 94.43%.

E. Various Predefined Models

In addition to the manually defined models, we also compared the performance of various predefined models. From the Keras pretrained vision nets [8] we implemented NASNetLarge [12], InceptionResNetV2 [13], DenseNet121,

and DenseNet201. NASNetLarge and InceptionResNetV2 were chosen because they were reported to have the highest accuracy according to the Keras documentation. In preliminary testing we also found DenseNet121 and DenseNet201 to work well with the dataset, so they were tested as well. Each model was trained on the 150x150 CNRPARK-EXT images using the arguments include_top=False, weights="ImageNet", and input_shape=(150,150,3) except for NASNetLarge which used input_shape=(331,331,3) and TensorFlow's Resizing layer. For each model, the trainable flag was set to False. Each model was then topped with a multilayer perceptron with two 128 node hidden layers. The Adam optimizer was used during training with the default learning rate. With the added multi-layer perceptron, the total parameters for each of these models comes to 147,381,332 for NASNetLarge, 56,123,106 for InceptionResNetV2, 22,271,042 for DenseNet201, and 9,151,554 for DenseNet121.

IV. Results

In this section, we present results on the CNRPark+EXT dataset from 10 different vision models.

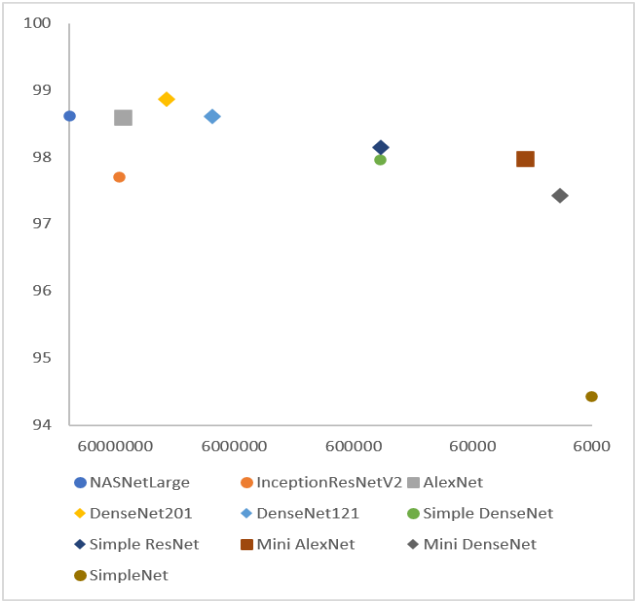

Figure 5: Accuracy percentage (vertical axis) vs. number of parameters (horizontal axis)

Model	Params	Train Acc	Test Acc
NASNetLarge	147,381,332	98.33%	98.62%
InceptionResNetV2	56,123,106	97.85%	97.71%
AlexNet	50,887,285	98.82%	98.59%
DenseNet201	22,271,042	98.18%	98.87%
DenseNet121	9,151,554	98.39%	98.61%
SimpleDenseNet	361,794	98.41%	97.97%
SimpleResNet	352,770	97.53%	98.15%
MiniAlexNet	21,628	97.58%	97.97%
MiniDenseNet	11,026	98.33%	97.43%
SimpleNet	6,082	92.25%	94.43%

Table 4: Model parameters and accuracy

A few trends can be observed in Figure 5 and Table 4. First, high accuracy can be achieved with relatively few parameters. The mDenseNet model we defined achieved 97.43% accuracy with only 11.026 parameters. Second, while increasing the number of parameters did generally increase accuracy, an additional 22,260,016 parameters were required to bring accuracy from 97.43% with the mDenseNet to 98.87% with DenseNet201. This is approximately a 0.064 percentage point increase per one million additional parameters. We tested 2 networks larger than DenseNet201: InceptionResNetV2 and NASNetLarge. These 2 networks were tested because they had the highest reported accuracy according to Keras's documentation [8]. Neither of these networks were more accurate than AlexNet or DenseNet201, despite their larger size. So, the two trends we observed were that a small, well implemented network could perform well on the parking problem and that increasing the size of these networks only marginally increased performance.

Figure 6: 3x3 descaling of 8 "Busy" images and 8 "Free" images

To investigate the nature of why, for this dataset, relatively simple networks are working almost as well as much more complex models, we downscaled the parking images to 3 by 3 pixels as seen in Figure 2. Looking at Figure 6 you can see that there is a general difference between the two classifications: the "Free" images appear flatter and more monochromatic. SimpleNet was implemented to determine how well the dataset could be predicted if all detail was removed and only the rough color gradient remained. This simple network still gives 94.43% accuracy on the testing set, which seems to suggest that the networks tested are perhaps not only learning what motor vehicles look like, rather, it may be that the networks are also looking for factors like the overall

discordance of the image or the range of colors present within the image.

V. Conclusions

We found that, at least with respect to the CNR+EXT dataset, it is possible to achieve remarkably high accuracy with minimal parameters using CNNs of various architectures. As such, we conclude that designing, implementing, and deploying a system to accurately and automatically classify the occupancy status of parking spaces in a parking lot using only a given image is well within the realm of possibility given current compute and Internet of Things capabilities. Using either the mAlexNet model or the mDenseNet would be sufficient to produce acceptable accuracy while minimizing the overall compute required. We recommend either of these models be used by a system that aims to convey the occupancy status of parking spaces to searching drivers.

VI. What did you learn

A. *Anthony Harris*

- Reading through the papers and learning about the different architectures used to improve CNN performance, both on a compute level and an accuracy level was probably the most valuable thing that I learned from this project. Between that and furthering my experience with the various tools provided by Tensorflow and Keras like Tensorboard and the plot_model util respectively, this has been an invaluable experience. There was also some need to focus on model pipeline optimizations as we were dealing with enough data that it wasn't always possible to keep everything in memory. Given my experience with, and the performance of, the DenseNet architecture, I have applied a variation based heavily upon the concatenated residual connections in my thesis research and intend to continue using it in the future.

B. *Joshua Ellis*

- I learned about the ImageDataGenerator object from TensorFlow. Data generators can be used to load and preprocess image datasets and stream them to the GPU if the dataset is too large.
- I learned about several new computer vision models. I found DenseNet to be the most interesting. Its ability to "short cut" its layers seems to have helped it in this dataset. I also tested Resnet50 at one point and it did not perform especially well for its size. I believe the mediocre performance of large Resnets was due to the depth of their layers making it such that they wanted to find more complex patters in the data. My intuition is that DenseNets gives them the ability to learn more complex patters when necessary, but also allows them to more easily fit their weights and biases to simpler patterns in the data.
- SimpleNet was an interesting experiment that suggests some of my intuitions about what the networks were learning. My intuition is that the networks are leaning what cars look like to some extent, but I think much of their predictions are also based on some of the broader patters in the images.

C. *Naseem Saquer*

- I learned about several new CNN architectures like the Densenet and the NASNet. I was most interested by the NASNet concept of creating an architecture on the fly based on the specific dataset in question. I also found the problem of being limited on the number of parameters you can use for a CNN based on the available hardware to be fascinating: even if a network like AlexNet performs especially well on a dataset, the high amount of RAM needed makes it an unideal solution given the constraints of the problem. Trying to optimize the tradeoff between accuracy and the number of parameters taught me the importance of making sure a network is minimal- this kind of idea had never crossed my mind before taking this class and doing this project.

References

[1] K. Aftab *et al*., "Reducing Parking Space Search Time and Environmental Impacts: A Technology Driven Smart Parking Case Study," in *IEEE Technology and Society Magazine*, vol. 39, no. 3, pp. 62-75, Sept. 2020, doi: 10.1109/MTS.2020.3012329.

[2] R. Mangiaracina, A. Tumino, G. Miragliotta, G. Salvadori and A. Perego, "Smart parking management in a smart city: Costs and benefits," *2017 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, Bari, 2017, pp. 27-32, doi: 10.1109/SOLI.2017.8120964.

[3] N. Bibi, M. N. Majid, H. Dawood and P. Guo, "Automatic Parking Space Detection System," 2017 2nd International Conference on Multimedia and Image Processing (ICMIP), Wuhan, 2017, pp. 11-15, doi: 10.1109/ICMIP.2017.4.

[4] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, "CNRPark+EXT A Dataset for Visual Occupancy Detection of Parking Lots," CNR Parking Dataset - Dataset for visual occupancy detection of parking lots. [Online]. Available: http://cnrpark.it/. [Accessed: 01-Dec-2020].

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017.

[6] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, "Car parking occupancy detection using smart camera networks and Deep Learning," 2016 IEEE Symposium on Computers and Communication (ISCC), 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[8] Team, K., 2020. Keras Documentation: Keras Applications. [online] Keras.io. Available at: <https://keras.io/api/applications/> [Accessed 3 December 2020].

[9] "The Functional API; TensorFlow Core," TensorFlow. [Online]. Available: https://www.tensorflow.org/guide/keras/functional. [Accessed: 01-Dec-2020].

[10] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[11] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, “Deep learning for decentralized parking lot occupancy detection,” Expert Systems with Applications, vol. 72, pp. 327–334, 2017.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning Transferable Architectures for Scalable Image Recognition,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the impact of residual connections on learning,” *arXiv* [cs.CV], 2016