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

**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

**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, 3]. Given the already challenging state of affairs 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 already overpopulated 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, 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 access 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 global 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 [2]. 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.

**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 Networks (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 [9]; 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 NUMBER\_ME, NUMBER\_ME\_TOO, and NUMBER\_ME\_THREE respectively.

  
*Figure NUMBER\_ME: Sample images from CNR-EXT [4, 9]*

|  |  |
| --- | --- |
| Class Frequencies | |
| Class | **Count** |
| Free | 65658 |
| Busy | 79307 |

*Table NUMBER\_ME*

|  |  |  |
| --- | --- | --- |
| Weather Frequencies | | |
| Weather Type | **Count** |
| Sunny | 63178 |
| Overcast | 44243 |
| Rainy | 37544 |

*Table NUMBER\_ME\_TOO*

|  |  |  |
| --- | --- | --- |
| Class Frequencies by Weather | | |
| Class | **Weather Type** | **Count** |
| Free | Sunny | 25665 |
| Overcast | 21067 |
| Rainy | 18926 |
| Busy | Sunny | 37513 |
| Overcast | 23176 |
| Rainy | 18618 |

*Table NUMBER\_ME\_THREE*

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.

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.

**Methods**

**AlexNet / Mini AlexNet**

One of the first network architectures that we applied to the CNRPark+EXT dataset was the Alexnet, which is one the most popular CNN architectures to date and was first proposed in [6]. The Alexnet is a very deep convolutional network with 51 million parameters. Despite its linear, relatively simple makeup seen in [FIGURE X], 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 ReLU activation function for non-linearity. The usage of ReLU for such large networks is notable, since it was found in [6] that CNNs could be trained much faster than when using saturating activations like sigmoid or 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 five convolutional layers in the model, with each layer being immediately followed by batch-normalization and max-pooling. Each fully connected layer utilizes 40% dropout in addition to the batch-normalization. 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 [6] 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.

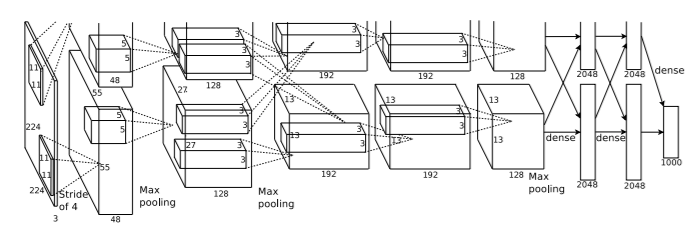
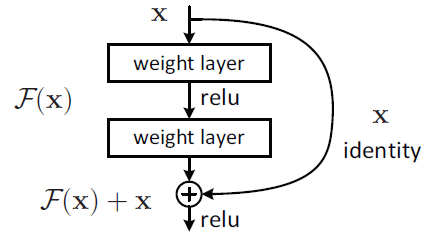


Figure NUMBER\_ME: Visualization of the Alexnet [6]

In terms of performance, the AlexNet achieved an accuracy of 98.59%. Relative to other architectures explained later in the paper, this

**SimpleResNet**

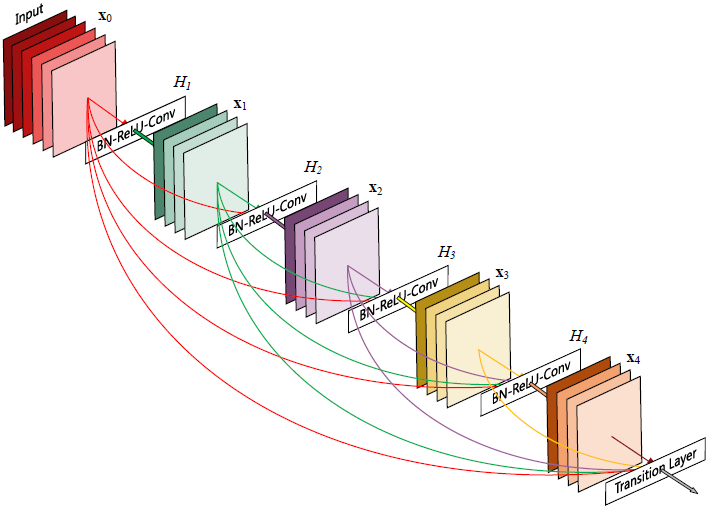
Given the success of the AlexNet and Mini AlexNet, we wondered how a more modern architecture like a ResNet [8] would perform. ResNets are unique in that they counter the exploding and vanishing gradient problems frequently encountered in deeper neural networks; they do so by designing residual connections where the output of a layer is added to the input of a layer some number of layers farther down the network architecture. Figure NUMBER\_ME shows an example of what this residual connection looks like graphically. While there are several existing ResNet model architectures provided by Keras and Tensorflow [12], we chose to build a small, simple ResNet model based upon the example given in the Tensorflow documentation at [10]. 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.



*Figure NUMBER\_ME: Example of a Residual Connection [8]*

**DenseNet / Mini DenseNet**

Like the ResNet, the DenseNet model architecture [7] 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 NUMBER\_ME 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 [7].

  
*Figure NUMBER\_ME: Example of a Dense Block [7]*

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, as can be seen in the Results section.

To further improve on the small parameter count and high accuracy achieved by the mini AlexNet from [4, 5], we decided to design and implement 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 only contains a total of 11,026 parameters, 10,866 of which are trainable. This is approximately 51% of the size of the mini AlexNet, and it performs comparably, as shown in the Results section.

**SimpleNet**

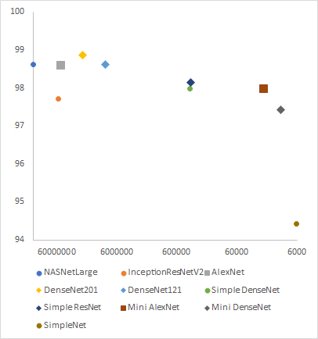
We implemented SimpleNet which takes the same 150x150 images as the other networks but SimpleNet uses average pooling with 50x50 pools to reduce the images to 3x3. Simple net then uses a multi-layer perceptron with two 64 node layers to classify the dataset. This model was implemented as an experiment to understand what the other models might be learning and to quantify the bassline 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.

**Various Predefined Models**

In this project we compared several models against the AlexNet and Mini AlexNet used in the original parking dataset papers. From the Keras pretrained vision nets [12] we implemented NASNetLarge, InceptionResNetV2, DenseNet121, and DenseNet201. NASNetLarge and InceptionResNetV2 were chose because they 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.

**Results**

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

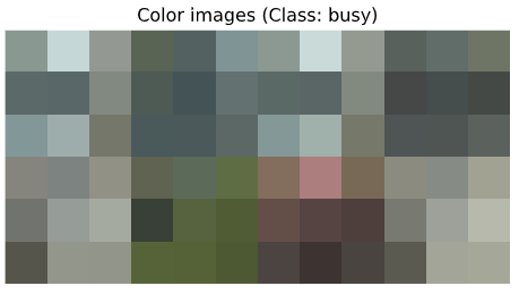


*Figure 1: Accuracy percentage (vertical axis) vs. number of parameters (horizontal axis)*

|  |  |  |  |
| --- | --- | --- | --- |
| **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% |
| Simple DenseNet | 361,794 | 98.41% | 97.97% |
| Simple ResNet | 352,770 | 97.53% | 98.15% |
| Mini AlexNet | 21,628 | 97.58% | 97.97% |
| Mini DenseNet | 11,026 | 98.33% | 97.43% |
| SimpleNet | 6,082 | 92.25% | 94.43% |

*Table 1: Model parameters and accuracy*

A few trends can be observed in Figure 1 and Table 1. First, reasonably high accuracy can be achieved with a relatively few parameters. The Mini DenseNet we put together 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 Mini DenseNet to 98.87% with DenseNet201. This is roughly a 0.064 percentage point increase per one million additional parameters. We tested two networks larger than DenseNet201, these were InceptionResNetV2 and NASNetLarge. These two networks were tested because they had the highest reported accuracy according to Keras’s documentation [12]. 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 2: 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 descaled the parking images to 3 by 3 pixels as seen in Figure 2. Looking at Figure 2 you can see that there is a general difference between the two classifications. The free images appear flatter and more monochromatic. Simple net 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. This seems to suggest that the networks tested are perhaps not only learning what cars look like. It may be that the networks are also looking for factors like the overall disformity of the image or the range of colors present within the image.

**Conclusion**

TODO: write

**What did you learn?**

* Anthony Haris
* 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.
* 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 a number of new computer vision models. I found DenseNet to be the most interesting.

**Citations**

[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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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].

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

[11] 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.

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

[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.