جامعة الأخوين

ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵓⵅⴰⵡⵉⵏ

AL AKHAWAYN UNIVERSITY

School of Science and Engineering

SMART PARKING SYSTEM USING MACHINE LEARNING MODELS

Capstone Design

Submitted in

April 2022

by

Mohamed Merrouch

Supervised by

Dr. Naeem Nisar Sheikh, Al Akhawayn University

Capstone Report

Student Statement:

I, Mohamed Merrouch, have applied ethics to the design process and in the selection of the final proposed design. I have held the safety of the public to be paramount and have addressed this in the presented design wherever may be applicable.

Mohamed Merrouch

Mohamed Merrouch

Approved by the Supervisor

Naeem Nisar Sheikh

Dr. Naeem Nisar Sheikh

Acknowledgments

I would like to start this report by thanking all the people who supported me throughout the process of accomplishing my capstone project and my journey at Al Akhawayn University.

Primarily, I would like to thank my supervisor. Dr. Naeem Nisar Sheikh who was there to support me throughout the Capstone process and trusted my abilities in completing it in the best way possible.

A special thank you goes to my friends who also encouraged me and supported me. They believed in my abilities even when I doubted them.

Besides, I would like to thank my professors at Al Akhawayn who prepared me to overcome any challenges in life.

Also, I would like to thank my family who sacrificed time and energy to allow me to be at Al Akhawayn University.

Finally, I want to gift this work to my grandfather Mohamed Merrouch who passed away last year. I made sure to inherit not only his name but also be inspired by his passion and morals throughout my stay at AUI and beyond.

Abstract

Smart Parking and Parking Guidance and Information systems are needed in today's big cities. It will reduce traffic, air pollution, and drivers stress and wasted time in finding a parking space. This report outline the design and implementation of a CNN model trained on the PKLot dataset to detect vacant parking spaces in parking lot images. The goal of this work is to build a robust model with increased generalization capabilities for it to be trained on PKLot but used on new parking lot camera feeds. This work achieved 0.9976 an accuracy of the model when trained on multiple parking lots. The model also achieved 0.9772 accuracy in average when trained on a single parking and testes on the other two parking lot camera feeds. The previous results show that this work could achieve robust generalization capabilities. These generalization capabilities allow the model to be deployed for commercial use in real-life parking lot management systems. It can also be a basis for future research covering such as transfer learning to new parking lots data or addition of more layers that will fine-tune the model and improve it.

Résumé

Les grandes villes d'aujourd'hui ont besoin de systèmes intelligents de guidage et d'information sur le stationnement. Ils réduiront le trafic, la pollution de l'air, le stress des conducteurs et le temps perdu à trouver une place de stationnement. Ce rapport décrit la conception et la mise en œuvre d'un modèle CNN entraîné sur le dataset PKLot pour détecter les places de parking inoccupées dans les images de parkings. L'objectif de ce travail est de construire un modèle robuste avec des capacités de généralisation accrues pour qu'il puisse être entraîné sur PKLot mais utilisé sur de nouveaux flux de caméras de parking. Ce travail a permis d'obtenir une précision de 0,9976 pour le modèle lorsqu'il a été entraîné sur plusieurs parkings. Le modèle a également obtenu une précision moyenne de 0,9772 lorsqu'il a été entraîné sur un seul parking et testé sur les deux autres alimentations de caméra de parking. Les résultats précédents montrent que ce travail peut atteindre des capacités de généralisation robustes. Ces capacités de généralisation permettent au modèle d'être déployé pour une utilisation commerciale dans des systèmes de gestion de parkings en conditions réelles. Il peut également servir de base à de futures recherches, telles que l'apprentissage par transfert sur de nouvelles données de parking ou l'ajout de couches supplémentaires qui affineront le modèle et l'amélioreront.

.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

AUI = Al Akhawayn University in Ifrane

CCTV = Closed-circuit television

CNN = Convolutional neural network

IoT = Internet of Things

JSON = JavaScript Object Notation

CNRST = Centre national de recherche scientific et technique

HPC = High performance computing

1. Introduction

Parking is becoming problematic in big cities all over the world. Morocco's busiest cities such as Casablanca started to show a need for such smart parking solutions. Moroccan statistics gathered by the Moroccan Ministry of transportation show that the country reached 4.06 million cars in 2017. Thus, an increase in Moroccan circulating cars by 18% from 2014 to 2017. Casablanca alone is containing 37% of circulating cars in Morocco [2].

Searching a parking spot causes stress, increases the pollution rate, contributes to traffic jams during peak hours, and wastes the driver's valuable time. It is estimated that drivers waste 7.8 minutes on average looking for parking spots. This represents 30% of the traffic flows in cities [3].

Smart parking advancements are either IoT solution designs or image processing solutions leveraging machine and deep learning. Using the latter techniques, we can solve multiple types of problems within smart parking while avoiding the high maintenance and cost required for IoT and sensor-based solutions [4].

This capstone design will focus on the problem of real-time vacancy detection. Through CCTV cameras, we can get a parking lot picture covering all the parking spaces in which we are interested. The current project will also require the bounding boxes of each parking space as an input. The aim is to design and implement a deep learning model that processes the cropped images of parking spaces from the image frame and predicts their occupancy as seen in Figure 1. The advantage of using CCTV cameras and a deep learning trained model comes in handy with huge parking lots. It reduces the amount of hardware and sensors needed by a sensor-based solution.

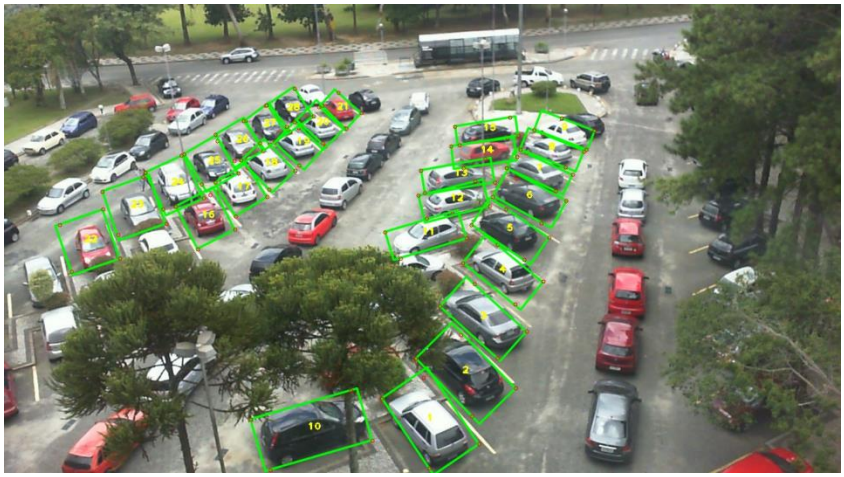


Figure 1: UFPR04 parking lot segmented with bounding boxes. Each parking space bounding box color reflects its occupancy status [5].

Previous related work has been done to achieve this objective with high accuracy and robustness. This project will build upon these findings and try to improve the already achieved results. The main contribution of this work will be leveraging the existing CNN deep learning architectures and tuning them to increase the generalization capabilities of the model. This report will investigate how reliable our model can be if trained on data from a parking lot and tested on different parking lots or camera angles.

To achieve this, we used a Convolutional Neural Network to extract features and classify the parking spaces. The generalization capabilities were integrated by using dropout layers, batch normalization layers, and

early stopping techniques while training the model. The dataset split - into training, validation, and test subsets - was prepared in a way to reduce any training bias.

The aim for high generalization capabilities is deemed important. It creates the potential for leveraging the robust parking dataset provided by Almeida et al. [5], to get to trained models with commercialization potential or use on university-owned parking lots such as the ones at our university, Al Akhawayn University in Ifrane.

The following sections will tackle the literature review, the feasibility study, the methods, and dataset used, the design and implementation details, the results, and the STEEPLE analysis of the project. Finally, we will be highlighting conclusions and suggest future work leads.

2. Feasibility Study

The purpose of this feasibility study is to analyze whether the capstone project scope is realistically accomplished within the given timeframe, computational resources, previously acquired knowledge, and previous academic work. To achieve this, the minimum project scope was refined further. The purpose is to design a deep learning solution that uses a CNN architecture that feeds on images of parking spaces. It then detects the occupancy status of each parking lot space. The study covers open-air parking lots during the daytime.

This project will go through six main phases. The first phase is the learning phase which is acquiring the necessary theoretical and technical knowledge to be able to analyze better previous academic work and implement later our model. This phase only requires time and access to scholarly articles as resources.

The second phase will be data collection. In the case of this project, we found multiple available datasets used by researchers. The third phase is data preparation. After going through the literature review and following the best practices acquired in the learning phase, we will be able to explore the dataset and pre-process it.

The next phase is the model design phase which will take more time. It will include designing and implementing the model architecture. It will need the practical knowledge acquired in the learning phase, foundational programming skills from previously taken courses, and the use of Keras and TensorFlow frameworks.

Once the model is designed, we will move to the training phase of the model on the data we prepared previously. High computational resources will be needed. Thus, acquired access to the pro version of the online cloud-based platform Collaboratory by Google. The latter provided us with access to a set of powerful CPUs, GPUs, and TPUs. We also managed to submit computational jobs to the CNRST HPC cluster infrastructure.

Finally, we will analyze and evaluate the results based on metrics we found in the literature review or from the learning phase to first fine-tune the model, and then finalize the project and author the report.

To conclude, any deep learning project will have to go through the above phases, and our project scope was defined in a way that accounts for the project deadline.

The project was successfully achieved through consistency, time management, and discipline. Table 1 shows the time distribution in percentages across the project phases mentioned above.

Phase description	Percentage of working time
Learning Phase and Literature Review	25%
Design of the complete pipeline from data input to evaluation metrics	19%
Dataset splits and data preprocessing	13%
Implementation of the model	13%
Training the model	13%
Evaluation of the model and analysis of the results	13%
Authoring the final report	6%

Table 1 Distribution of work on this project across the project phases

3. Literature Review

Vacancy detection systems are an important part of PGI systems (Parking Guidance and Information. The latter are classified based on the detection method or data input. The first type is counter-based systems. They rely on sensors at the entrance and exit of parking lots. They only provide information on the number of spots available in a parking lot. This means that the parking users do not get any further guidance on where the spaces are available. Furthermore, these systems are not compatible with on-street parking spaces, or residential parking spaces. The second type is wireless or wired sensor-based systems. They provide a high degree of reliability. They are applied in in-door parking lots such as shopping malls. This type is based on installing sensors in each parking space. The costs of sensors, transceivers, and processing units, and the high maintenance needs of this kind of vacancy detection system affect its affordability and scalability. This especially applies to huge open-air parking lots with many parking spaces to set up. Our project falls under the third category of PGI systems: camera-based vacancy detection. The latter is less costly compared to using sensors since CCTV cameras cover multiple parking spaces with one single angle in open-air parking lots. Also, the same cameras are used for both vacancy detection, and general security and surveillance [6].

Early related work within vacancy detection designs based on camera inputs focused on image processing, handcrafted feature extractions, and training of machine learning classifiers on these extracted features. Funck et al. started by comparing reference images with input images using principal component analysis as a basis for the detection [7]. Next works in line focused on training classifiers such as Bayesian, SVM, and Random Forest classifiers along with a multitude of image processing techniques to extract image features such as corners, edges, texture, colors, and background subtraction. The feature extraction techniques used were SIFT, ORB, BRISK, and others [6][8].

Since these methods were all based on manually engineered feature extraction, the accuracies were not higher than 90% and were highly perturbed by the change in weather conditions. The systems based on convolutional neural networks improved these results.

Valipour et al. used the VGGNet-F pre-trained model on ILSVRC-2012. The authors then retrained the last layers on the Almeida et al. dataset PKLot as well to fine-tune it. Furthermore, this work goes into the details of the choice of the pre-trained model and the methods used in finetuning it. The authors used an SGD optimizer with a 0.01 learning rate. The network was trained for three thousand iterations with 128 samples per mini-batches. The work also provides a system architecture for the use of such a model in production by developing a user interface to set the bounding boxes of parking spaces and monitor real-time these parking lots [8].

Acharya et. al built upon Valipour et al.'s work, among others. The authors used the same PKLot dataset along with a pre-trained CNN to extract features. The innovation in this work lies in using a trained SVM as a classifier and getting the final binary output. This paper also does transfer learning on the street where the researchers live, and it is promising. The CNN-SVM combination was interesting in the sense that it complemented the weaknesses of both when used separately [6].

Nyambal et al. also used fine-tuned pre-trained CNN. The contribution of their work is providing the process of collecting a new dataset of parking spaces images. Thus, it helps understand better how the dataset PKLot can be leveraged. The paper also compares the use of three different solvers when training the model [9].

Cazamias et al. worked on parking space vacancy detection within the scope of their class project. The authors also used the PKLot dataset. They used a CNN with a simple architecture compared to the pre-trained CNNs mentioned above. Their CNN architecture had three convolutional layers with 10, 20, and 30 filters, respectively. The kernel size they used was five pixels by five. They also used batch normalization layers after each convolutional layer. They used max-pooling layers to reduce the dimensionality at the end of the three convolutional layers. Finally, they used three fully connected layers to classify the input. The paper achieved 0.9997 accuracy on the test set. They also trained and tested the model on the three different parking lot camera feeds of the dataset to evaluate the generalization of their model [10]. The results are shown in Figure 2.

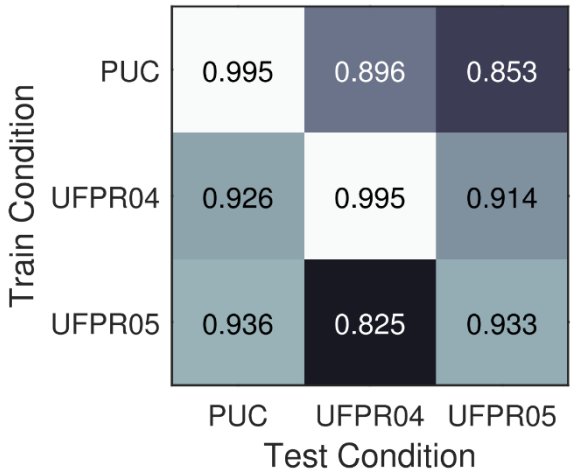


Figure 2: Confusion matrices for Cazamias et al.'s binary classifier based on parking lot camera feeds. For each result, the CNN was trained on the condition of row i, then tested on the condition of column j [10].

The last paper worth mentioning is the paper related and provided with the dataset we are using in this project. Almeida et al. gathered the PKLot dataset and provided with it a description of data collection methods, pre-processing, the guidance on how to constitute the training and testing sets, suggestions on evaluations metrics, and potential research directions [5].

To summarize, the existing state-of-art works use CNN architectures to solve the problem of parking vacancy detection. Some make use of fine-tuned pre-trained models, while others either create their own architecture or use different classifiers. The PKLot dataset is the backbone dataset for vacancy detection research in the literature. In addition, many researchers collect datasets themselves to examine the transfer learning capabilities of models trained on the PKLot dataset. In this project, we will also use Almeida et al.'s dataset to design a robust and generalizable CNN model that can be used on other parking lot camera feeds without having to retrain it.

4. Method

4.1. Convolutional Neural Networks

Computer vision problems were solved using traditional image processing and machine learning classification algorithms. CNN came later when more processing power and data became more available for research. CNN are nowadays used now for a variety of applications such as facial recognition, healthcare, self-driving cars, accessibility, and biology.

Computer vision is founded on the ability to detect low dimension features from a high-dimensional input image. Previously, computer vision researchers had to master the domain knowledge depending on the task, define these features manually, and then detect these features using different classifiers. Humans' ability to identify these features is limited because of the variations that affect a 3D object in the world such as scale, viewpoints, Illumination, occlusion, deformations, background clutter, intra-class variations, and others. Figure 3 shows an example of such variations [14].

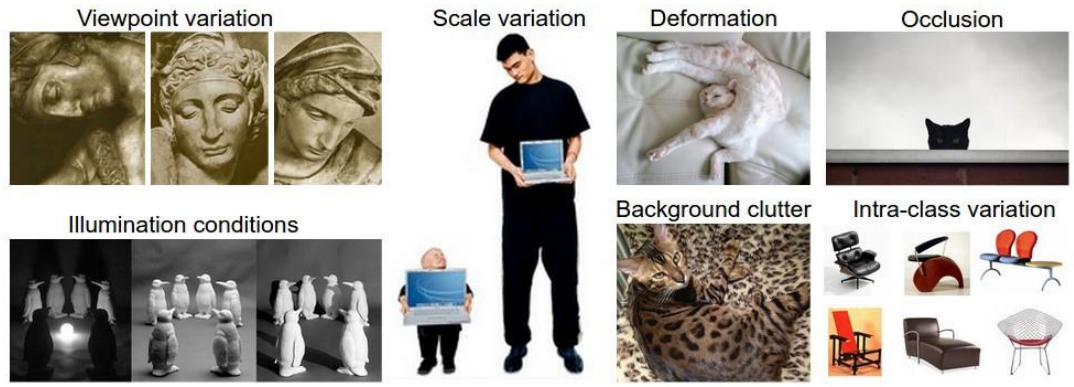


Figure 3: The different image variation that are considered as a challenge in front of manual feature extraction from images [11].

CNN uses neural networks to perform automatic feature extraction. Through this architecture, we can learn a hierarchy of features directly from the large datasets of images. An example of such a hierarchy of features is shown in Figure 4.

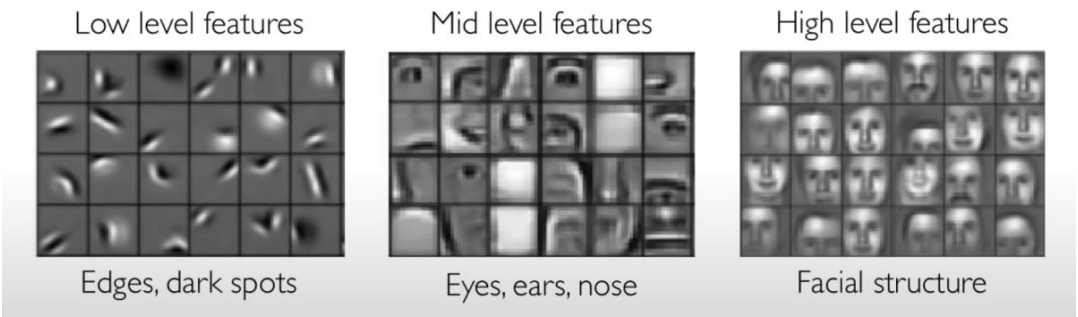


Figure 4: Example for a hierarchy of features that are detected by a CNN [11].

Given that an image representation is usually a 3D matrix of numbers, the use of regular ANNs through fully connected layers that will take pixel values of these matrices as an input will result in parameter explosions and the loss of the spatial information of the image.

Therefore, CNN is based on the use of a 2D sliding patch over the input images and provides outputs to the neurons that will make the new extracted feature in 2D as well. This will maintain the spatial information as shown in Figure 5. This patching operation is based on the convolution mathematical operation. More specifically, it is sliding a weighted matrix, doing element-wise multiplication, and adding up the result. The weighted matrix is called a filter with weights that are learned in the training. The output of the convolution layer over the input image is called a feature [14].

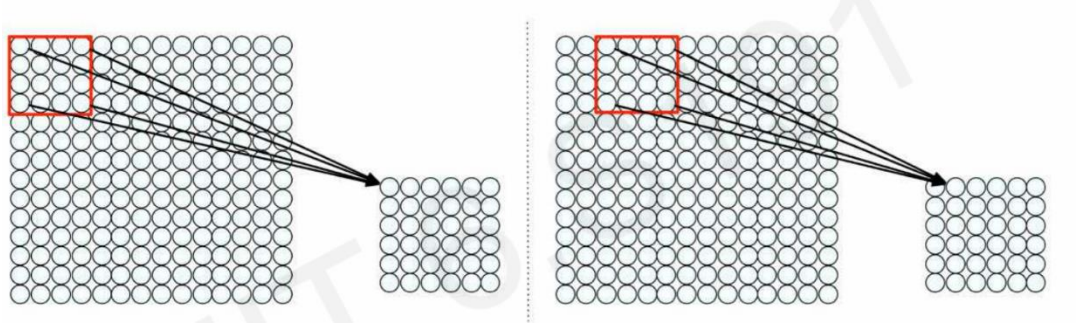


Figure 5: The sliding patch over the image and outputting another 2D weighted feature [14].

The CNN is thus designed based on three building blocks. First, are the convolution layers that generate feature volumes. The convolutional layer can have multiple filters with each filter producing a different feature map. The feature maps are stacked together to produce a feature volume that is next forwarded to the next layer. The second building block is applying non-linearity to reflect the non-linear nature of real life. ReLu function is the most common activation function used in CNN. Finally, the feature volumes are processed by pooling layers. The pooling layers are used to downsample the features and reduce dimensionality. This allows keeping the scalability of the input and spatial invariance and structure [14]. This process is shown in Figure 6.

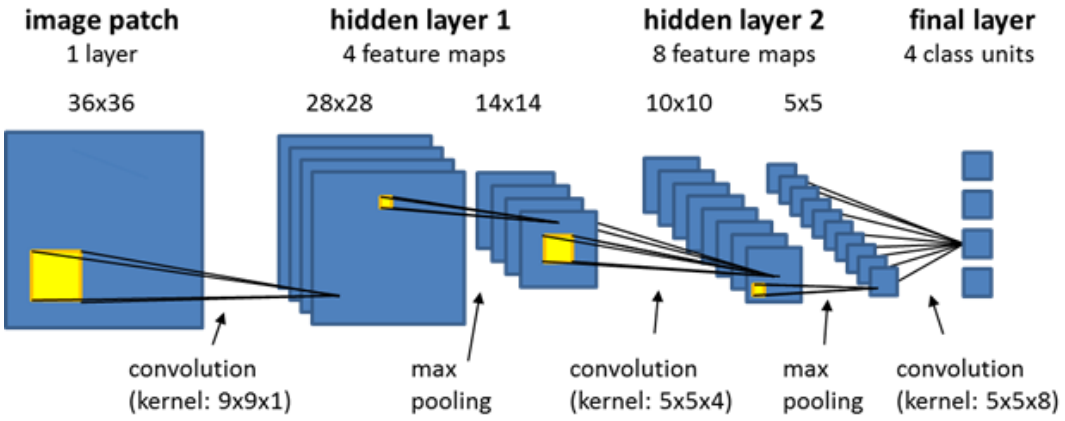


Figure 6: An example of convolution layers and maxpooling layers acting on an input image [12].

These three building blocks' key role is feature learning and extraction. Next, the output is forwarded to fully connected layers to perform the classification task [14] as shown in Figure 7.

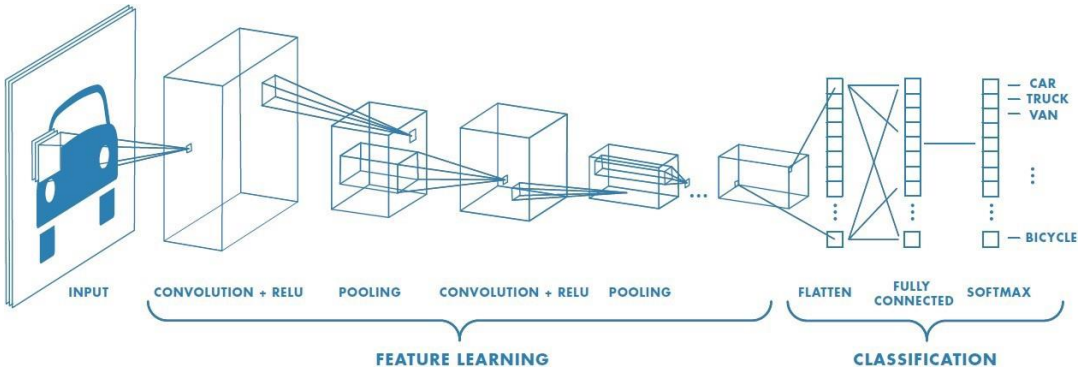


Figure 7: A complete CNN architecture with both the feature learning first and the classifier next [13].

The CNN predictive power is increased in the training through using a loss function, and backpropagation to update the weights and minimize the loss [10].

4.2. Dropout Layers

To achieve good generalization capabilities of the model, we need to make sure the model is not overfitting the training dataset. Dropout is a regularization technique that helps prevent or reduce overfitting. The idea of the dropout layer is to switch off the activation of a percentage of neurons in a layer by setting their activation to zero. The percentage is advised to be 50% or less to not break the learning process. These neurons are sampled randomly at each training pass. This forces the model to not rely on single nodes and unravel different paths while learning weights. The dropout layers are advised to be put after the dense layers and not convolution layers [14] [15].

4.3. Batch Normalization

Batch Normalization is a normalization technique done on the inputs of each layer separately. The normalization is calculated for the mini-batches separately. This technique calculates the mean and variance of each mini-batch to normalize it. Then the normalized value is scaled by a parameter Beta and shifted by a parameter Lambda. Both Beta and Lambda are learned to not lose the expressiveness of the model [10][14]. The algorithm is shown in Figure 8.

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

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

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

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

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

Figure 8: Batch Normalization algorithm [10].

The added value of batch normalization in our model architecture is to speed up the training process since the inputs are normalized. The normalized inputs and outputs allow layers to learn independently. This technique also eliminates the problem of the initial initialization of the neural network that affects the performance each time we initialize differently. Finally, and most importantly, batch normalization can also serve as a regularization actor and avoid the overfitting of the model. The regularization effect comes from the fact that the normalization by mini batch adds some noise each time to the model's data distribution. Batch normalization is usually combined with dropout layers since the added noise is small [10][14][15].

4.4. Early Stopping

The model also overfits due to the high number of epochs done on the dataset. The more we train the model on the dataset the more it starts creating one-to-one mappings between the training data samples and the desired labels to minimize the loss. This reduces the predictive power of the model and pushes it to be weak at predicting the labels of samples that were not part of the training sub-set. Figure 9 shows an example of overfitting [14].

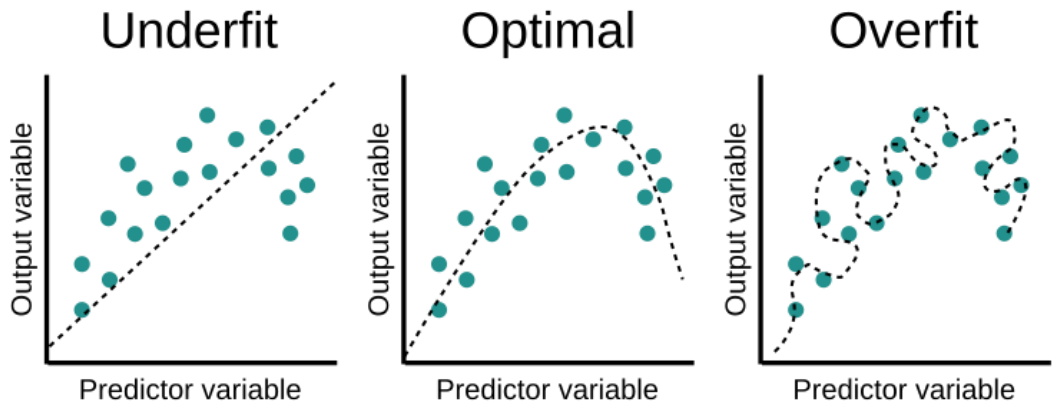


Figure 9: Example of underfitted model, robust model, and overfitted model of 2D sample data [16].

The early stopping technique is also a regularization technique that helps identify the right moment when to stop the training. The idea is to train the model on training data and minimize the training loss while monitoring

the loss of the model on a validation subset of the dataset that is used for training or testing. The loss quantities both reduce in the first phase of the training, then they start diverging when the model overfits and start accumulating losses on the validation set. The perfect moment for an early stop is when the training loss and validation loss start diverging [14] as shown in Figure 10.

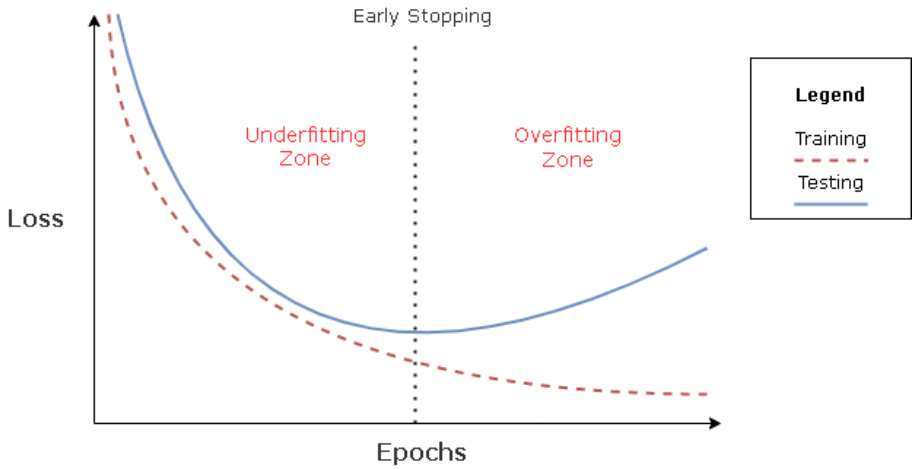


Figure 10: Training example illustrating the underfitting and overfitting regions and the optimal epoch to stop [17].

4.5. Adam Optimizer and Learning Rate Decay

The CNN performs a forward pass through the neurons of the neural network. Then a backpropagation pass is done to compute the gradients of all the trainable neural network parameters over the loss function used. The next step is to update the model's weight in a direction that will minimize the loss. This updating step is decided by a learning rate that is set as a hyperparameter at the beginning of the training. The choice of this learning rate affects the training. A small learning rate can end up with the training stuck at a local minima, while a big learning rate can make the training chaotic and unable to find a minima [14]. The Adam optimizer has an algorithm that allows the learning rate to adapt to the specificities of the loss function. This helps also in increasing the training speed [10]. Also, combining the Adam optimizer with an inverse time learning decay allows the training to go faster as well. The inverse time decay we used "hyperbolically decreases the learning rate to 1/2 of the base rate at [5] epochs, 1/3 at [10] epochs, and so on" [18].

5. Dataset

The dataset we used to train our model is the PKLot dataset collected by Almeida et al. using three different camera feeds and two parking lots. The dataset includes pictures from three distinct kinds of weather: rainy, sunny, and cloudy.

The dataset has 12417 taken images in total. It also provides 695851 segmented and cropped images of individual parking spaces extracted from the parking lot images. The skew adjustment was performed on all segmented pictures to eliminate perspective distortion and they were all oriented vertically as shown in Figure 11. The segmented images are manually labeled with either 'Empty' or 'Occupied.' The images have varied sizes [5]. Table 2 shows a summary of segmented images used from the dataset.

Figure 11: Examples of occupied and empty spots in the dataset [10].

Parking Lot	# Parking spaces per parking	# of segmented empty parking space images	# of segmented occupied parking space images	Total segmented parking space image
UFPR04	28	46,125	59,720	105,845
UFPR05	45	97,426	68,359	165,785
PUC	100	194,229	230,040	424,269
Total	173	337,780	358,119	695,899

Table 2 Summary of the PKLot segmented images characteristics

6. Implementation

In this work, we are building up on the previous works through designing and implementing a CNN architecture for our model to predict the occupancy status of parking spaces. We investigated two research directions suggested in Almeida et al. dataset paper. The first one is training our model on multiple parking lots at once. This will help evaluate how our model's ability to absorb the variability in camera viewpoints, surface patterns, and camera mounting height. The second research direction is training our model on one parking and evaluating it on the other parking lots not accessible in training subsets. The latter will show us how generalizable our model when trained on one single parking. The following sub-sections will tackle the architecture of our CNN model, the data preparation, the training method, and the technical approach.

6.1. Architecture of the CNN model

Our model is a fine-tuned version of the model suggested by Cazamias et al. in their work. We used three convolution layers with a Relu activation function. The three layers have 10, 20, and 30 filters respectively with a kernel size of 5x5. We added a batch normalization after each convolution layer along with a ReLu activation function. Then, we added max pooling layers of filter size of 2x2 to the second and third convolution layers. Finally, we flattened the output of the convolutional layers to feed it to the next three fully-connected layers with 128, 32, and 16 neurons, respectively. The first and second dense layers were followed by dropout layers with 50%, and 10% dropout rate each. Finally, we used a Softmax Layer with two outputs to compute the class scores [10]. The model architecture is illustrated in Figure 12 while Table 3 provides a more detailed summary of the model.

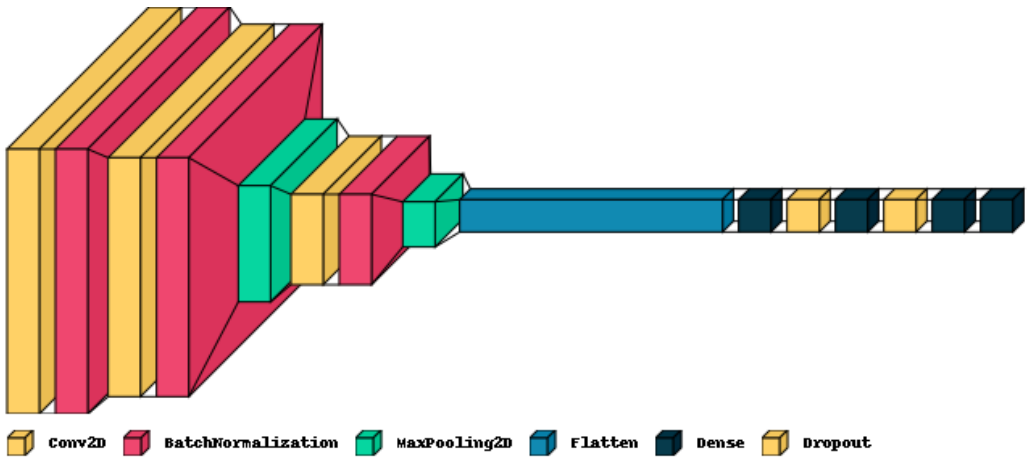


Figure 12: Diagram of our CNN architecture for classifying the parking space occupancy.

Layer type	Output Shape	Parameters number
Conv2D	(Batch_size, 66, 41, 10)	760
BatchNormalization	(Batch_size, 66, 41, 10)	40
Conv2D	(Batch_size, 62, 37, 20)	5,020
BatchNormalization	(Batch_size, 62, 37, 20)	80
MaxPooling2D	(Batch_size, 31, 18, 20)	0
Conv2D	(Batch_size, 27, 14, 30)	15,030
BatchNormalization	(Batch_size, 27, 14, 30)	120
MaxPooling2D	(Batch_size, 13, 7, 30)	0
Flatten	(Batch_size, 2730)	0
Dense	(Batch_size, 128)	349,568
Dropout	(Batch_size, 128)	0
Dense	(Batch_size, 32)	4,128
Dropout	(Batch_size, 32)	0
Dense	(Batch_size, 16)	528
Dense	(Batch_size, 2)	34
Total Parameters		375,308
Trainable Parameters		375,188
Non-Trainable Parameters		120

Table 3 Our CNN model summary showing details of each layer in terms of type, output shape, and number of parameters

6.2. Data Preparation

Definition of Training/Validation/Testing sets

Almeida et al. suggests in their paper to split the data into 50% training subset and 50% testing subset. According to the same paper, the main guiding principle to follow is not having samples of the same parking lot at the same day in both training, validation, and testing subsets. This will bias the results since the model may be trained and evaluated in this case on the same cars that stayed for a long time of the day in a parking lot. The only difference would be light variations which will confuse the training of the model [5].

In our work, we made sure to follow this rule while also leveraging the large amount of data available in the PKLot dataset to also have a validation set. The validation set is crucial for our early stopping mechanism and the increase of the generalization capabilities of the model. The original folder structure of the PKLot dataset is shown in Figure 13.

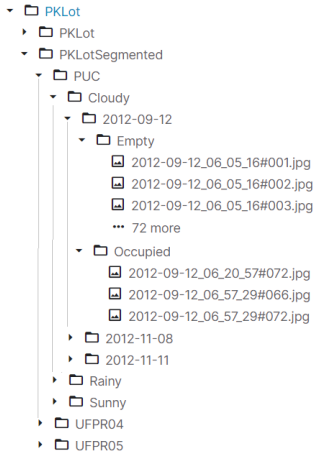


Figure 13: Original folder structure of the segmented images folder in the PKLot Dataset

We extracted the file paths of all segmented images and reorganize the data following the JSON representation shown in Figure 14. Our suggested structure allows us to prioritize the class name (i.e., occupancy status), to make sure we have balanced training/validation/testing subsets in terms of class names. The second advantage is keeping the separation of the three parking lot camera feeds to abstract the splitting of each camera feed from the other. Finally, and most importantly, we stored the file paths of segmented parking spaces by dates under each parking lot camera feed. This gave us control over creating a clear separation of training/validation/testing by dates as well. Hence, we could respect the rule set by Almeida et al. [5] without affecting our ability in splitting the dataset in more than two parts. This method is promising as it can quickly scale up to apply the k-fold cross-validation approach without stepping over the day separation rule.

```
{
  "empty": {
    "puc": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    },
    "ufpr04": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    },
    "ufpr05": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    }
  },
  "occupied": {
    "puc": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    },
    "ufpr04": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    },
    "ufpr05": {
      "date_1": ["path_1", "path_2"],
      ..............................
      "date_n": ["path_n", "path_m"]
    }
}
```

Figure 14: The JSON representation of our choice of data structure schema used to store the dataset image paths

For the multiple parking training and testing, we split the dataset into ~50% samples for training, ~10% for validation, and ~40% for testing.

For the single parking training and multiple parking testing, we split each parking samples separately into ~70% training, ~10% validation, and ~20% testing. Since our goal from training on single parking lots is to evaluate and boost the generalization capability, we made sure to use a validation set that aggregates the samples from the three parking lot camera feeds. Hence, we concatenated the validation sets of each parking lot camera feed together and use the joined set for the validation of single parking trainings. Table 4 shows a detailed summary of our training, validation, and testing sets.

		Training				Testing				Validation			
		Empty	Occupied	Total	Percentage	Empty	Occupied	Total	Percentage	Empty	Occupied	Total	Percentage
Single Parking Training	UFPR04	43,497	32,521	76,018	72%	6,445	7,221	13,666	13%	9,776	6,383	16,159	15%
	UFPR05	50,459	70,163	120,622	73%	5,091	9,179	14,270	9%	18,084	12,809	30,893	19%
	PUC	167,033	139,963	306,996	72%	20,534	22,151	42,685	10%	42,427	32,115	74,542	18%
Multiple Parking Training		189,556	178,745	368,301	53%	32,566	30,240	62806	9%	135,949	128,795	264744	38%

Table 4 Summary of training, validation, and testing sets

Definition of Training/Validation/Testing sets

As mentioned in section 5, the segmented images of the parking spaces in the dataset were already adjusted and oriented vertically. The only preprocessing we had to perform on the dataset is resizing all images to 70x45 as shown in Figure 15. This size was chosen based on computing the average size of the complete dataset. Then we normalized the range of pixel intensity values to a [0-1] range.

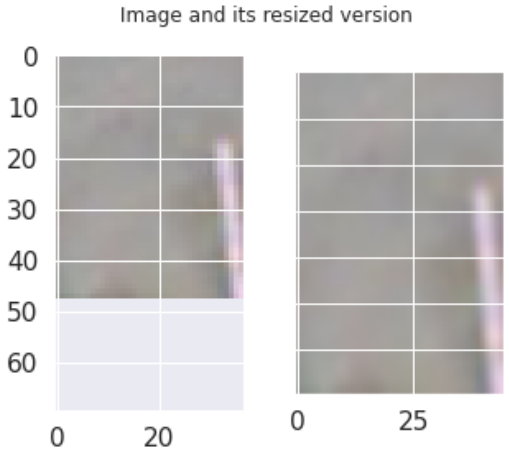


Figure 15: Example of an image from the dataset and its resized version

We included these preprocessing steps inside our custom Keras data generator. The latter receives the set of paths to include in the data generator, splits them into batches, reads the images of each batch, preprocesses them, and provides them to the model for either training, validation, or testing [19][20]. The complete code of our data generator is included in Appendix A.

6.3. Training method

We initialized all the different data generators required for all training experiments as shown in Figure 16. Then we evaluated the model on a subset of one hundred samples of the training data to make sure that the model can overfit the data. A model that cannot overfit a small subset of the dataset is unable to learn from the larger dataset [19].

```
[ ] #[MP_MT] Multiple parking training multiple parking testing data generators
    MP_MT_training_generator = DataGenerator(ALL_ds_split, 'training')
    MP_MT_validation_generator = DataGenerator(ALL_ds_split, 'validation')
    MP_MT_test_generator = DataGenerator(ALL_ds_split, 'test', shuffle=False)
```

```
[ ] #[SP_MT] Single parking training multiple parking testing data generators

    #PUC
    SP_MT_training_generator_PUC = DataGenerator(PUC_ds_split, 'training')
    SP_MT_test_generator_PUC = DataGenerator(PUC_ds_split, 'test', shuffle=False)

    #UFPR04
    SP_MT_training_generator_UFPR04 = DataGenerator(UFPR04_ds_split, 'training')
    SP_MT_test_generator_UFPR04 = DataGenerator(UFPR04_ds_split, 'test', shuffle=False)

    #UFPR04
    SP_MT_training_generator_UFPR05 = DataGenerator(UFPR05_ds_split, 'training')
    SP_MT_test_generator_UFPR05 = DataGenerator(UFPR05_ds_split, 'test', shuffle=False)

    # This validation takes the validation data of the three parkings at once to be used
    # In all the three single parking trainings
    SP_MT_validation_generator = DataGenerator(unified_validation_ds_split, 'validation')
```

Figure 16: The initialization of all the data generators needed for the different trainings of the model.

Once the model was checked, we performed the following training methodology four times. Once for the multiple parking lot training set mentioned in section 6.2.1. We did the three remaining trainings on the training set of each parking lot camera feed separately. Table 5 provides the details of each training experiment separately.

Experiment	Number of training samples	Number of iterations per Epoch	Stopped at Epoch	Best weights at Epoch	Training Time per Epoch
Single parking training on UFPR04	51786	2877	13	5	~488s [slower GPU]
Single parking training on UFPR05	16956	942	12	4	~420s
Single parking training on PUC	43164	2398	13	5	~750s
Multiple parking training	51786	2877	19	11	~860s

Table 5 A summary of the four training experiments performed on the model

Compiling the model

The model was compiled using the Adam optimizer, with a learning rate of 0.001, a learning rate decay scheduler detailed in section 3.5, and a categorical cross entropy loss function.

Setting Callbacks

We defined multiple callbacks to be able to optimize the training process. We used first the early stopping callback and set it to monitor the validation loss. The training automatically stops after 8 epochs if the validation loss does not decrease further. This allowed us to minimize training time. The second callback used was a checkpoint callback that saves the weights that yields the best validation loss during the training time. This checkpoint was used later to load the weights and get the best model that avoids overfitting as explained in section 3.4. We also added a callback that logs the training progress and different metrics such as training loss and accuracy, validation loss and accuracy, and learning rate changes.

Training the model

We trained the model for a maximum of twenty-five epochs per training with mini-batches size of 128 samples. At each epoch, the validation set is also used to compute the validation loss and accuracy. All training experiments stopped before reaching twenty-five epochs. The training outputs the loss and accuracy of both training and validation of each epoch. This allowed us to plot charts such as the one in Figure 17 and Figure 18. The charts allowed us to analyze the training progress.

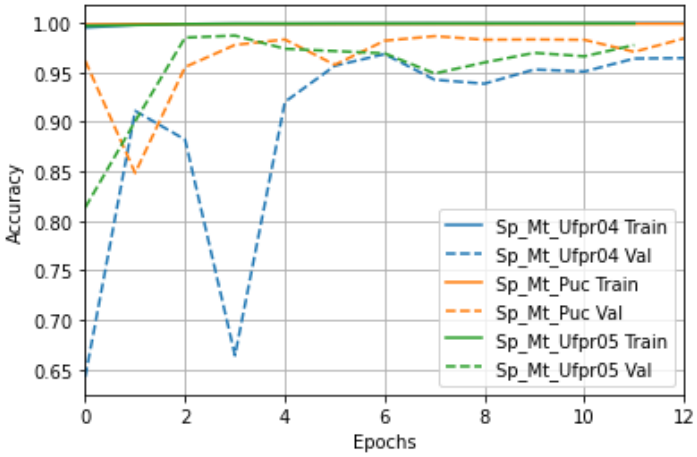


Figure 17: Accuracy vs epoch number when training single parking camera feeds.

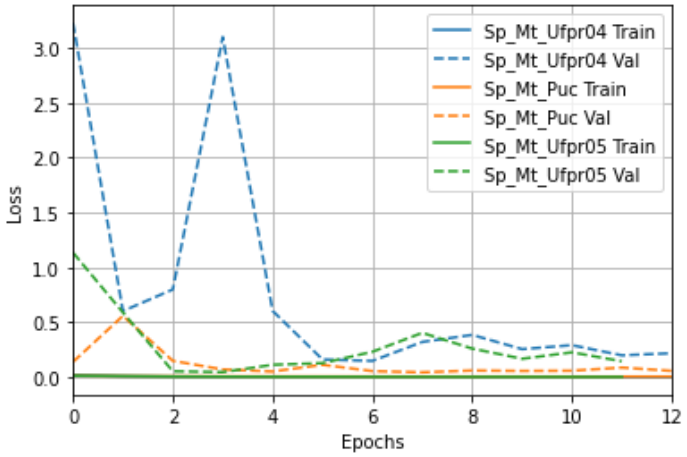


Figure 18: Loss vs epoch number when training single parking camera feeds.

Saving the models

Once the training was completed, we used the saved best weight checkpoint to save the complete model in HDF5 format. We also save the final model as well in the same format. These formats are reliably loaded later to evaluate and test them.

6.4. Technical approach

The above implementation was done using the second version of TensorFlow and its Keras API. The code was written and run on Collaboratory by Google. We used the Pro version of the platform to have access to larger storage, more stable runtime environment, a faster CPU, and NVIDIA Tesla 4 GPU. The main helper python modules were numpy, matplotlib, skimage, and glob.

7. Results

7.1. Evaluation Criteria

The main evaluation criteria suggested by Almeida et al. are confusion matrices and the AUC (Area Under Curve) of Receiver Operating Characteristics (ROC) the metric [5]. We also added the accuracy, precision, recall, and F1-Score. The latter metrics of all testing experiments can be found in Appendix B.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 19: Equations to calculate accuracy, precision, recall, and f1-score [21].

For the confusion matrix, our definition of the four quadrants is the following:

- TP is when the actual parking space is empty, and the model predicted that it is empty
- TN is when the actual parking space is occupied, and the model predicted that it is occupied
- FN is when the actual parking space is empty, and the model predicted that it is occupied
- FP is when the actual parking space is occupied, and the model predicted that it is empty

The ROC is created by plotting the TP rate against the FP rate at different threshold settings. A test is 100% accurate if the AUC of the ROC is one.

We used the metrics module of sklearn to calculate the above metrics.

In the case of parking space vacancy detection, both FN and FP need to be improved depending on the context. Decreasing FN would maximize parking owner profits for example through attracting the highest number of customers possible. Decreasing FP on the other hand will minimize the driver frustration of finding a crowded parking with no available spots.

7.2. Performance of the model

The model trained on multiple parking lot camera feeds reached an accuracy of 0.9976 which competes with all papers mentioned in section 2. The AUC of the ROC score was 0.99968. The ROC curve is shown in Figure 20 and the confusion matrix is shown in Figure 21.

Figure 20: The ROC of the testing experiment of the model trained on multiple parking.

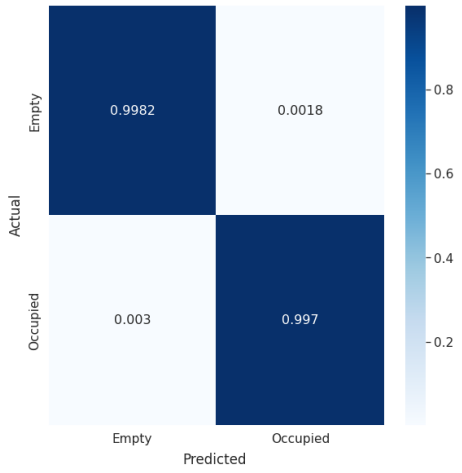


Figure 21: Confusion matrix of the testing experiment of the model trained on multiple parking.

This confirms the ability of our model to absorb the variability in camera angles, mounting height, weather conditions, and surface texture.

This model shows also shows robustness against overfitting. The testing accuracy only decreased by 0.0006.

7.3. Generalization Capabilities

We conclude that the model outperformed the results of Cazamias et al. when comparing Figure 2 with Figure 22 that shows the confusion matrix of accuracies [10]. The AUC of ROC score was also higher than 0.99480. Appendix B has details of all evaluation results. Table 6 provides a summary of accuracies and ROC AUC score of the ten testing experiments conducted in this work.

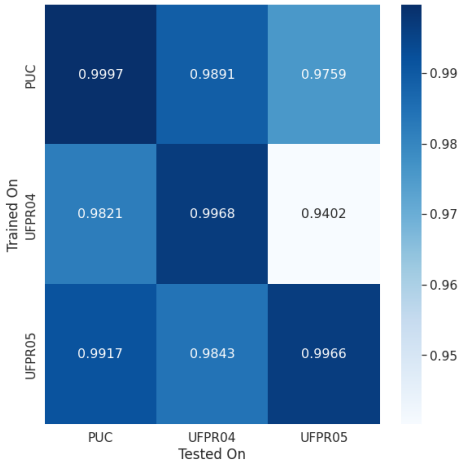


Figure 22: Confusion matrices for our binary classifier based on parking lot camera feeds. For each result, the CNN was trained on the condition of row i, then tested on the condition of column j.

We conclude that the designed model, data splits method, and training method detailed in section 6 increased the generalization capabilities of the based model designed by Cazamias et al. [10].

Testing Experiments			
Trained on	Tested on	Accuracy	ROC AUC Score
UFPR04	UFPR04	99.68%	99.99%
	UFPR05	94.03%	99.48%
	PUC	98.21%	99.88%
UFPR05	UFPR04	98.44%	99.87%
	UFPR05	99.65%	99.98%
	PUC	99.18%	99.95%
PUC	UFPR04	98.91%	99.95%
	UFPR05	97.58%	99.51%
	PUC	99.97%	99.99%
Multiple Parking Lots	Multiple Parking Lots	99.760%	99.968%

Table 6 Summary of the ten testing experiments conducted

8. Conclusion

Aiming to increase the generalization capabilities of our designed CNN model will allow parking vacancy detection research to leverage models trained on large datasets to be used reliably on new parking lots.

Our CNN model could achieve reliable results and outperform the generalization capabilities of previous works in the literature. The least accuracy achieved when training on a parking and testing on another was 0.9402 while the average accuracy was 0.9772 for the six different testing experiments.

These results were achieved through the addition of batch normalization layers, dropout layers, and the early stopping mechanism in the training method. The method used to split the dataset and extract a validation set as well made the training process more robust against overfitting as well.

Furthermore, the robustness of the PKLot dataset is the backbone of this project. The dataset key strong points were the considerable number of samples, the variations in weather conditions, the surface texture, the camera mounting height, the camera angles, and the slight light conditions.

The technical approach was smoothened through leveraging the power of the Keras API along with implementing our own custom data generator that helped speed up the training time.

This project can be further investigated from many aspects as we will detail in the next section. The generalization capabilities will allow the model to be deployed in new parking lots. This holds a potential for commercialization if combined with a user-friendly system, and robust architecture.

9. Future Work

9.1. Spatial Transformation Layers

As mentioned by Cazamias et al. as well, the accuracy drops when training the model on UFPR04 and UFPR05 could be minimized if we add a spatial transformation layer. This layer will "learn how to perform an affine transform on the feature map and therefore becomes invariant to rotations" [10] in the case of parking lots with different camera angles.

9.2. Interactive tool to set bounding boxes

The transfer learning of this model to new parking lots could be done on video stream coming from stationary cameras available on the internet or at the University. A tool that takes the first frame and provides and interactive interface to the user to set the bounding boxes of each parking space will allow the deployment and scalability of the current trained model.

9.3. Collecting a new dataset for transfer learning

Both Acharya et al. and Nyambal et al. were able in their works to collect their own dataset and perform transfer learning of the PKLot trained model on it [6][9]. We can also do the same using camera feeds of AUI different parking lots and cameras. This will allow us to evaluate the generalization capabilities on new parking lot with relatively smaller number of samples.

9.4. Using a pre-trained CNN and comparing results

Previous works in the literature used pre-trained CNN models and fine-tuned them. This allowed them to have stronger feature extractions and avoid starting near a local minima. A continuation of this work could be using a pre-trained CNN model and compare the generalization capabilities found in section 7.3.

References

[1] “How STEEPLE Analysis Informs Design Strategy,” *Bresslergroup*. [Online]. Available: https://www.bresslergroup.com/blog/design-defined-how-steeple-analysis-informs-design-strategy/. [Accessed: Apr. 17, 2022]

[2] “Plus de 4 millions de véhicules circulent officiellement au Maroc,” *Médias24*, Apr. 11, 2019. [Online]. Available: https://medias24.com/2019/04/11/plus-de-4-millions-de-vehicules-circulent-officiellement-au-maroc/. [Accessed: Apr. 17, 2022]

[3] R. Arnott and E. Inci, “An integrated model of downtown parking and traffic congestion,” *Journal of Urban Economics*, vol. 60, no. 3, pp. 418–442, Nov. 2006, doi: 10.1016/j.jue.2006.04.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0094119006000386. [Accessed: Apr. 17, 2022]

[4] Jo˜ao Victor Baggio, Luis Fernando Gonzalez, and Juliana Freitag Borin, “SmartParking - A smart solution using DeepLearning,” *Projeto Final de Graduação*, vol. Relatório Técnico-IC-PFG-20-10, Jul. 2020 [Online]. Available: https://smartcampus.prefeitura.unicamp.br/pub/artigos_relatorios/PFG_Joao_Victor_Estacionamento_Inteligente.pdf

[5] P. R. L. de Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich, “PKLot – A robust dataset for parking lot classification,” *Expert Systems with Applications*, vol. 42, no. 11, pp. 4937–4949, Jul. 2015, doi: 10.1016/j.eswa.2015.02.009. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0957417415001086. [Accessed: Apr. 17, 2022]

[6] D. Acharya, W. Yan, and K. Khoshelham, “Real-time image-based parking occupancy detection using deep learning,” in *Proceedings of the 5th Annual Conference of Research@Locate*, Adelaide, Australia, 2018, vol. 2087, pp. 33–40 [Online]. Available: http://ceur-ws.org/Vol-2087/paper5.pdf

[7] S. Funck, N. Mohler, and W. Oertel, “Determining car-park occupancy from single images,” in *IEEE Intelligent Vehicles Symposium, 2004*, Parma, Italy, 2004, pp. 325–328, doi: 10.1109/IVS.2004.1336403 [Online]. Available: https://ieeexplore.ieee.org/document/1336403/. [Accessed: Apr. 17, 2022]

[8] S. Valipour, M. Siam, E. Stroulia, and M. Jagersand, “Parking-stall vacancy indicator system, based on deep convolutional neural networks,” in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, Reston, VA, USA, Dec. 2016, pp. 655–660, doi: 10.1109/WF-IoT.2016.7845408 [Online]. Available: http://ieeexplore.ieee.org/document/7845408/. [Accessed: Apr. 17, 2022]

[9] J. Nyambal and R. Klein, “Automated parking space detection using convolutional neural networks,” in *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics*

(PRASA-RobMech), Bloemfontein, South Africa, Nov. 2017, pp. 1–6, doi: 10.1109/RoboMech.2017.8261114 [Online]. Available: http://ieeexplore.ieee.org/document/8261114/. [Accessed: Apr. 17, 2022]

[10] J. Cazamias and M. Marek, "Parking Space Classification using Convoluional Neural Networks," Stanford University, Course Project Report [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/280_Report.pdf

[11] K. Melcher, "Introduction to Convolutional Neural Networks and Computer Vision," *Low Code for Advanced Data Science*, Aug. 20, 2021. [Online]. Available: https://medium.com/low-code-for-advanced-data-science/introduction-to-convolutional-neural-networks-and-computer-vision-72b2d85dd1c0. [Accessed: Apr. 17, 2022]

[12] "Using Deep Learning Models / Convolutional Neural Networks." [Online]. Available: https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm. [Accessed: Apr. 17, 2022]

[13] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," *Medium*, Dec. 17, 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: Apr. 17, 2022]

[14] M. D. Learning, "MIT Deep Learning 6.S191," *MIT Deep Learning 6.S191*. [Online]. Available: http://introtodeeplearning.com. [Accessed: Apr. 17, 2022]

[15] "Everything You Should Know About Dropouts And BatchNormalization In CNN," *Analytics India Magazine*, Sep. 14, 2020. [Online]. Available: https://analyticsindiamag.com/everything-you-should-know-about-dropouts-and-batchnormalization-in-cnn/. [Accessed: Apr. 17, 2022]

[16] "Overfitting and underfitting," *Educative: Interactive Courses for Software Developers*. [Online]. Available: https://www.educative.io/edpresso/overfitting-and-underfitting. [Accessed: Apr. 17, 2022]

[17] A. Igareta, "The Million-Dollar Question: When to Stop Training your Deep Learning Model," *Medium*, Jun. 24, 2021. [Online]. Available: https://towardsdatascience.com/the-million-dollar-question-when-to-stop-training-deep-learning-models-fa9b488ac04d. [Accessed: Apr. 17, 2022]

[18] "Overfit and underfit | TensorFlow Core," *TensorFlow*. [Online]. Available: https://tensorflow.google.cn/tutorials/keras/overfit_and_underfit. [Accessed: Apr. 17, 2022]

[19] T. Dayanand, "Data Preprocessing and Network Building in CNN," *Medium*, Aug. 24, 2020. [Online]. Available: https://towardsdatascience.com/data-preprocessing-and-network-building-in-cnn-15624ef3a28b. [Accessed: Apr. 18, 2022]

[20] “A detailed example of data generators with Keras.” [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly. [Accessed: Apr. 18, 2022]

[21] admin, “An Introduction to Accuracy, Precision, Recall & F1-Score in Machine Learning - Machine Learning Tutorial,” *Tutorial Example*, Jan. 11, 2022. [Online]. Available: https://www.tutorialexample.com/an-introduction-to-accuracy-precision-recall-f1-score-in-machine-learning-machine-learning-tutorial/. [Accessed: Apr. 18, 2022]

Appendix A

```
class DataGenerator(tf.keras.utils.Sequence):
    'Generates data for Keras'

    def __init__(self, ds_split, mode='training', ablation=None, occupa
ncy_cls=['Empty', 'Occupied'],
                 batch_size=128, dim=(70, 45), n_channels=3, shuffle=Tr
ue):
        """
        Initialise the data generator
        """
        self.dim = dim
        self.batch_size = batch_size
        self.labels = {}
        self.list_IDs = []

        # glob through directory of each class
        #for i, cls in enumerate(occupancy_cls):
        self.list_IDs = ds_split["paths"][mode]
        self.labels.update({p:ds_split["labels"][mode][i] for i,p in en
umerate(ds_split["paths"][mode])})
        if ablation is not None:
            self.list_IDs = random.sample(ds_split["paths"][mode], len(
ds_split["paths"][mode]))
            self.list_IDs = self.list_IDs[:ablation]

        self.n_channels = n_channels
        self.n_classes = len(occupancy_cls)
        self.shuffle = shuffle
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.list_IDs) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.bat
ch_size]

        # Find list of IDs
        list_IDs_temp = [self.list_IDs[k] for k in indexes]

        # Generate data
        X, y = self.__data_generation(list_IDs_temp)
```

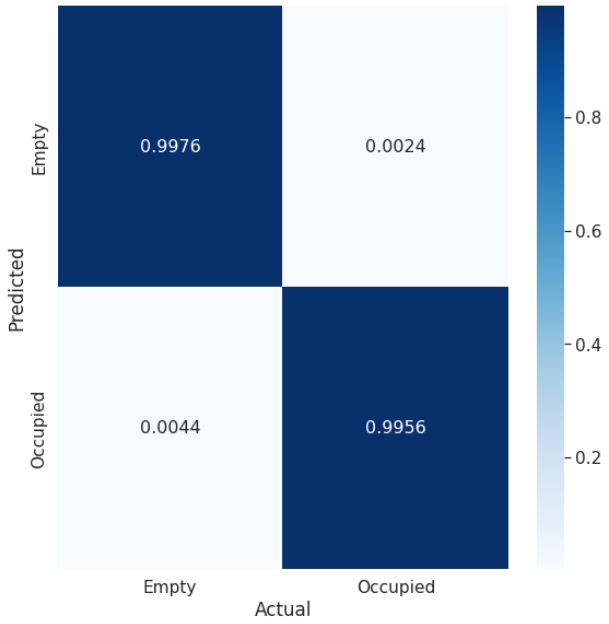
```
        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.list_IDs))
        if self.shuffle == True:
            np.random.shuffle(self.indexes)

    def __data_generation(self, list_IDs_temp):
        'Generates data containing batch_size samples' # X : (n_samples
, *dim, n_channels)
        # Initialization
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.empty((self.batch_size), dtype=int)

        delete_rows = []

        # Generate data
        for i, ID in enumerate(list_IDs_temp):
            # Store sample
            img = io.imread(ID)
            img = resize(img, (70, 45, 3))

            X[i,] = img

            # Store class
            y[i] = self.labels[ID]
        return X, tf.keras.utils.to_categorical(y, num_classes=self.n_c
lasses)
```

Appendix B

Trained on UFPR04 and tested on UFPR04

The scores of this model are 0.0106 for the loss and with an accuracy of 0.9968.

The ROC AUC score is: 0.99987

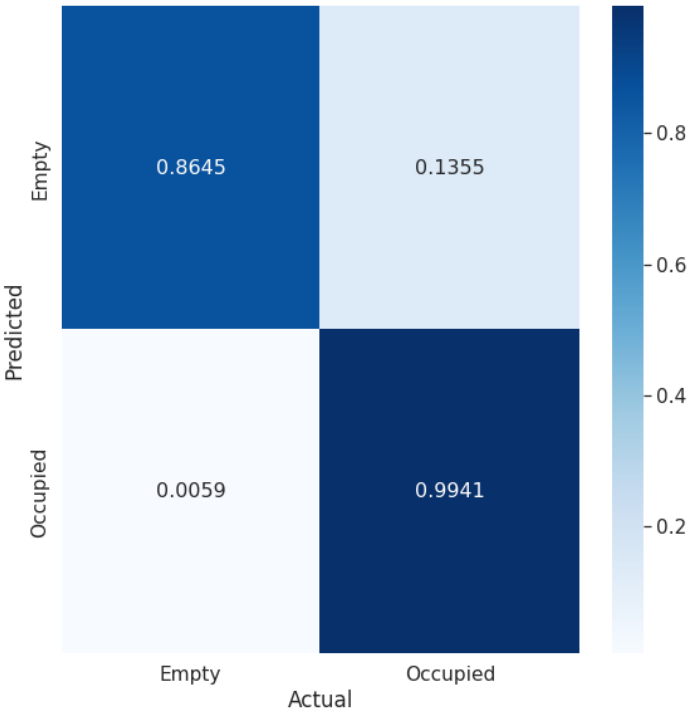


	precision	recall	f1-score	support
Empty	0.9971	0.9976	0.9974	9776
Occupied	0.9964	0.9956	0.9960	6352
accuracy			0.9968	16128
macro avg	0.9968	0.9966	0.9967	16128
weighted avg	0.9968	0.9968	0.9968	16128

Trained on UFPR04 and tested on UFPR05

The scores of this model are 0.2003 for the loss and with an accuracy of 0.9403.

The ROC AUC score is: 0.99480

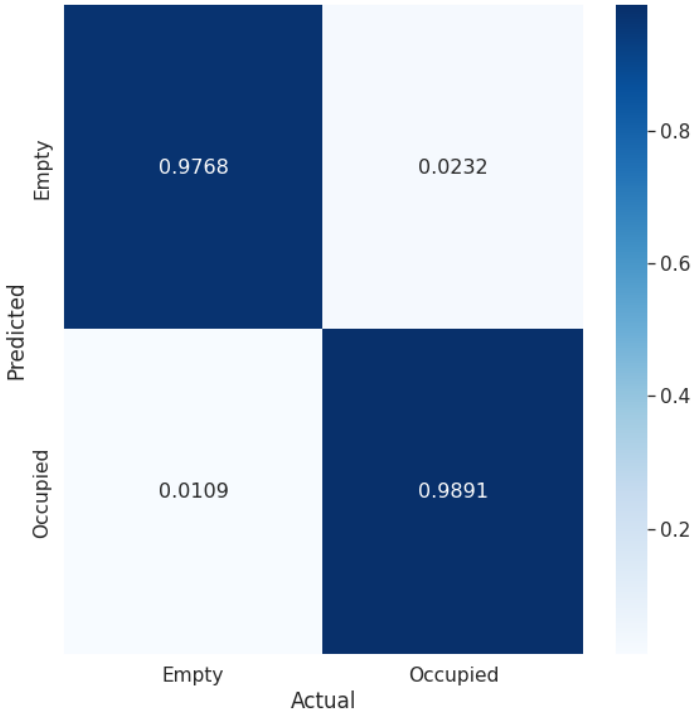


	precision	recall	f1-score	support
Empty	0.9904	0.8645	0.9232	12809
Occupied	0.9117	0.9941	0.9511	18039
accuracy			0.9403	30848
macro avg	0.9511	0.9293	0.9371	30848
weighted avg	0.9444	0.9403	0.9395	30848

Trained on UFPR04 and tested on PUC

The scores of this model are 0.0556 for the loss and with an accuracy of 0.9821.

The ROC AUC score is: 0.99875

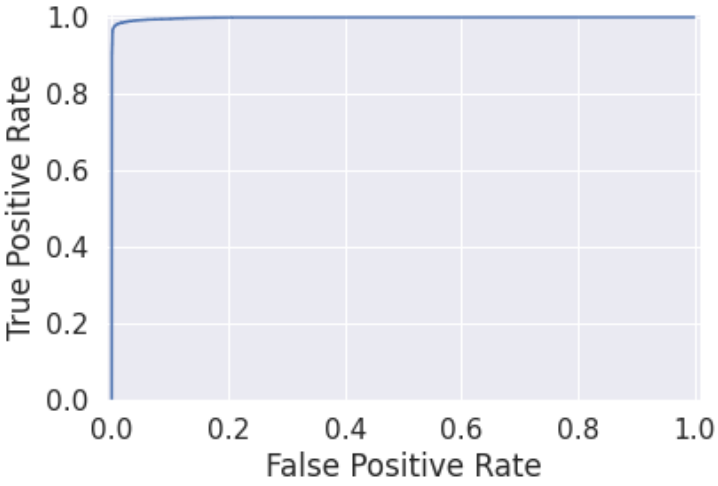


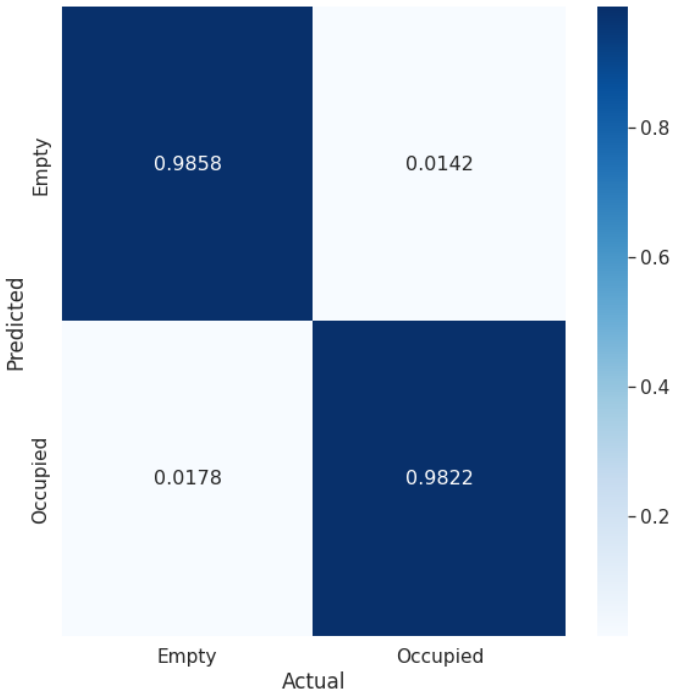
	precision	recall	f1-score	support
Empty	0.9917	0.9768	0.9842	42427
Occupied	0.9699	0.9891	0.9794	32069
accuracy			0.9821	74496
macro avg	0.9808	0.9830	0.9818	74496
weighted avg	0.9823	0.9821	0.9821	74496

Trained on UFPR05 and tested on UFPR04

The scores of this model are 0.0477 for the loss and with an accuracy of 0.9844.

The ROC AUC score is: 0.99867



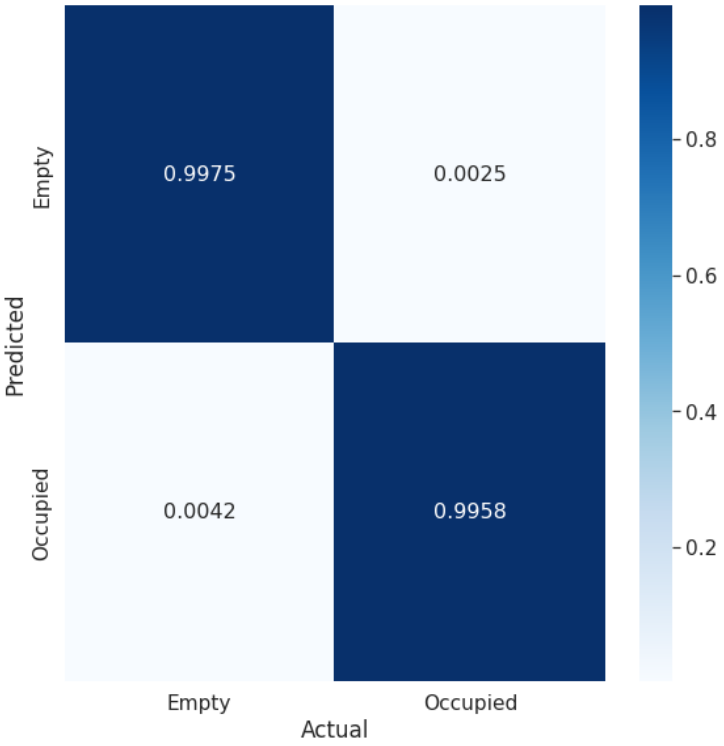


	precision	recall	f1-score	support
Empty	0.9884	0.9858	0.9871	9776
Occupied	0.9782	0.9822	0.9802	6352
accuracy			0.9844	16128
macro avg	0.9833	0.9840	0.9836	16128
weighted avg	0.9844	0.9844	0.9844	16128

Trained on UFPR05 and tested on UFPR05

The scores of this model are 0.0125 for the loss and with an accuracy of 0.9965.

The ROC AUC score is: 0.99984

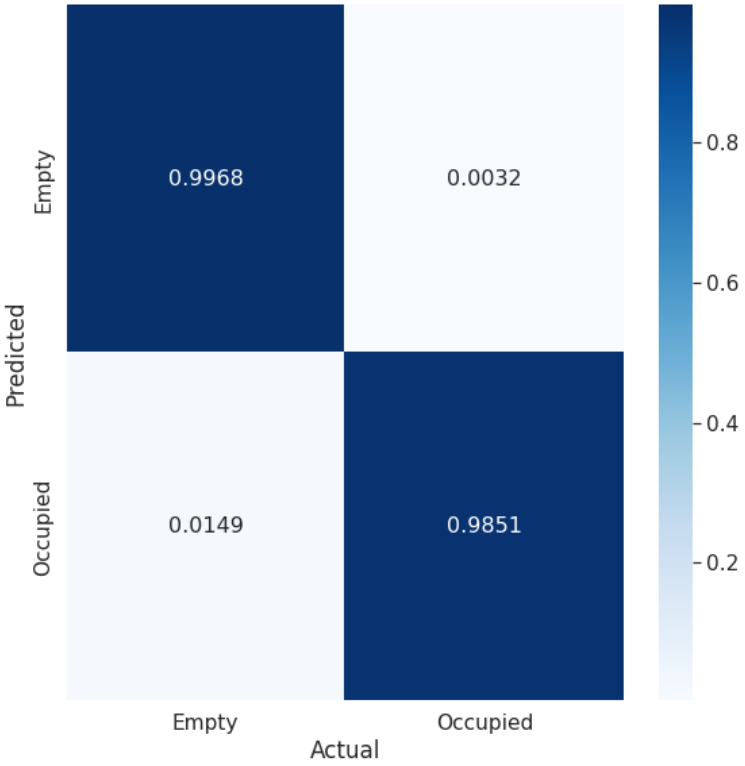


	precision	recall	f1-score	support
Empty	0.9942	0.9975	0.9958	12809
Occupied	0.9982	0.9958	0.9970	18039
accuracy			0.9965	30848
macro avg	0.9962	0.9967	0.9964	30848
weighted avg	0.9965	0.9965	0.9965	30848

Trained on UFPR05 and tested on PUC

The scores of this model are 0.0331 for the loss and with an accuracy of 0.9918.

The ROC AUC score is: 0.99951

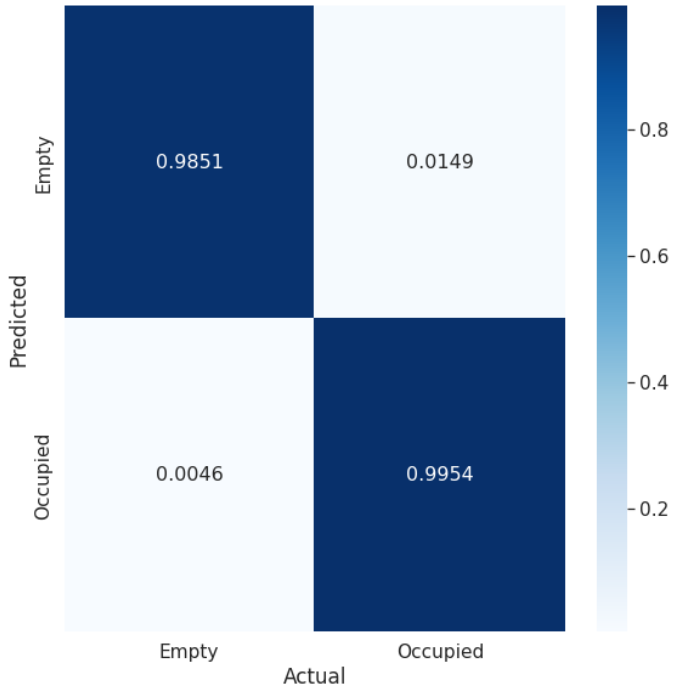


	precision	recall	f1-score	support
Empty	0.9888	0.9968	0.9928	42427
Occupied	0.9957	0.9851	0.9904	32069
accuracy			0.9918	74496
macro avg	0.9923	0.9909	0.9916	74496
weighted avg	0.9918	0.9918	0.9918	74496

Trained on PUC and tested on UFPR04

The scores of this model are 0.0364 for the loss and with an accuracy of 0.9891.

The ROC AUC score is: 0.99949

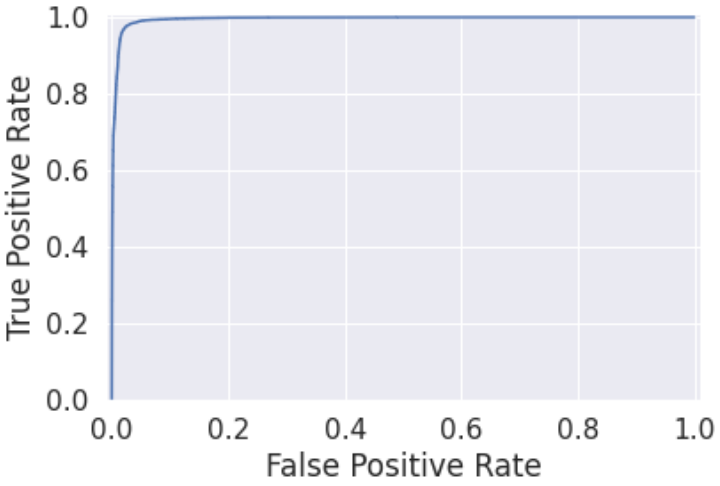


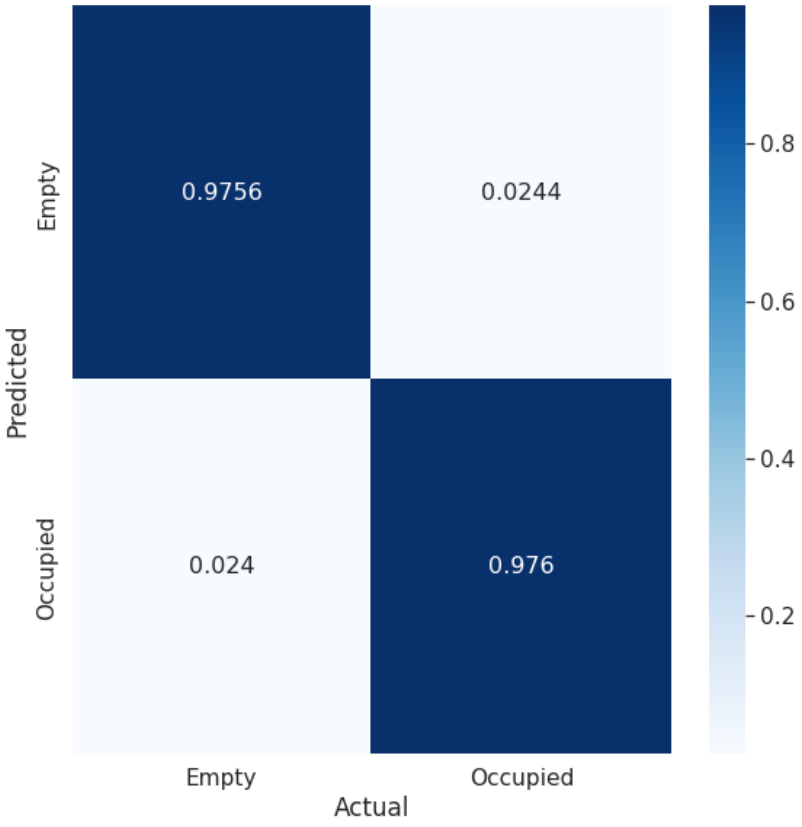
	precision	recall	f1-score	support
Empty	0.9970	0.9851	0.9910	9776
Occupied	0.9774	0.9954	0.9864	6352
accuracy			0.9891	16128
macro avg	0.9872	0.9902	0.9887	16128
weighted avg	0.9893	0.9891	0.9892	16128

Trained on PUC and tested on UFPR05

The scores of this model are 0.0775 for the loss and with an accuracy of 0.9758.

The ROC AUC score is: 0.99512



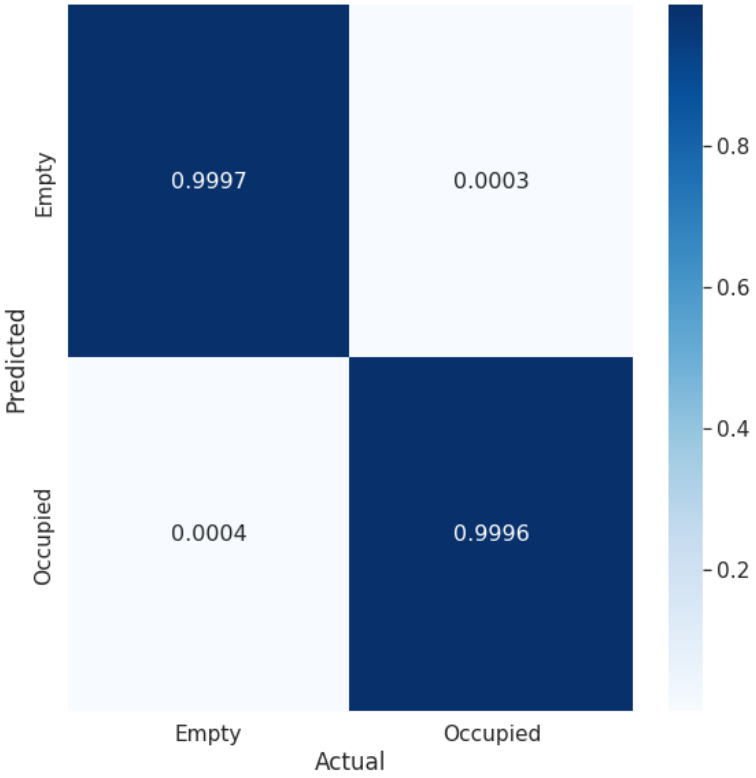


	precision	recall	f1-score	support
Empty	0.9665	0.9756	0.9710	12809
Occupied	0.9825	0.9760	0.9793	18039
accuracy			0.9758	30848
macro avg	0.9745	0.9758	0.9751	30848
weighted avg	0.9759	0.9758	0.9758	30848

Trained on PUC and tested on PUC

The scores of this model are 0.0033 for the loss and with an accuracy of 0.9997.

The ROC AUC score is: 0.99989

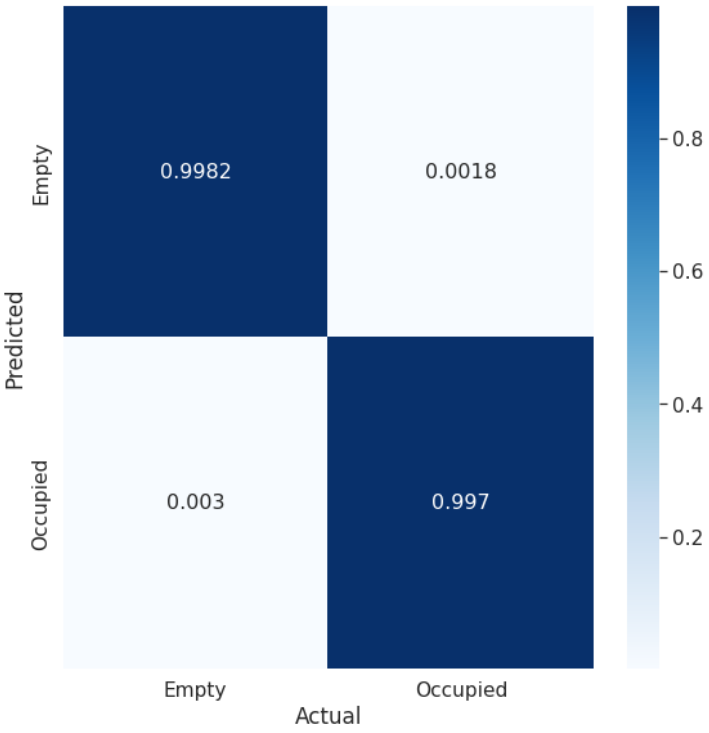


	precision	recall	f1-score	support
Empty	0.9997	0.9997	0.9997	42427
Occupied	0.9996	0.9996	0.9996	32069
accuracy			0.9997	74496
macro avg	0.9997	0.9997	0.9997	74496
weighted avg	0.9997	0.9997	0.9997	74496

Trained on multiple parking lots and tested on multiple parking lots

The scores of this model are 0.0109 for the loss and with an accuracy of 0.9976.

The ROC AUC score is: 0.99968



	precision	recall	f1-score	support
Empty	0.9971	0.9982	0.9977	135949
Occupied	0.9981	0.9970	0.9975	128755
accuracy			0.9976	264704
macro avg	0.9976	0.9976	0.9976	264704
weighted avg	0.9976	0.9976	0.9976	264704