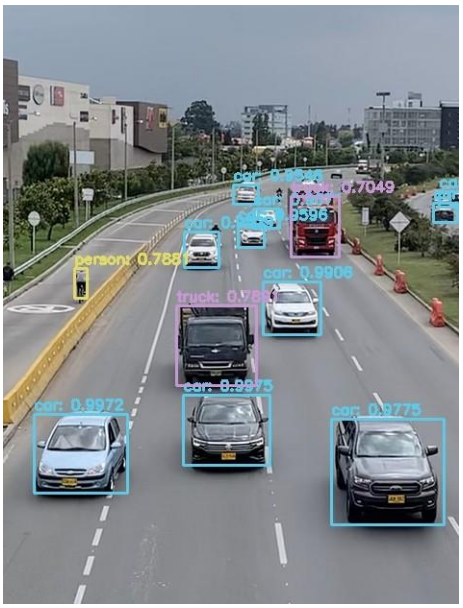

Computer Vision for Object Detection

TRAFFIC PATTERN MODELLING APPLICATION

Embry-Riddle Aeronautical University

Honors Program

Department of Mathematics

Presented to: Professor Steven Lehr

Jose Nicolas Gachancipa

Scientific Computing (MA305)

Introduction

Computer vision is a subset of artificial intelligence which involves the processing and analysis of digital images. Computer vision systems aim to simulate the behavior and capabilities of the human visual system (Huang, 1996). The development and applications of computer vision models have become widely popular in the past decade, due to advancements in computing capabilities and performance. Computer vision applications cover a wide variety of industries and are used for tasks such as object detection and recognition, defect detection in manufacturing, and face verification.

One of the most common models used in computer vision is the convolutional neural network (CNN), which is a type of artificial neural network. CNNs, also known as ConvNets, were specifically designed for image analysis (Thanki & Borra, 2019) and use feature extraction to identify objects in digital images. The purpose of the project is to explore the use of CNNs for traffic pattern modelling using a security camera, Python, TensorFlow and Google Colab. The model developed uses CNNs to detect and classify objects in camera recordings. The model identifies the traffic patterns (throughout the day) of objects such as cars, motorcycles, bicycles, and people. The software implementation can be found in the following Github repository:

https://github.com/nicolasgapa/Traffic-Modelling-using-Computer-Vision-CNNs-

Background

Convolutional Neural Networks

Convolutional Neural Network (CNNs or ConvNets) were first developed in the early 1990s. As most subsets of Artificial Intelligence, computer vision models are inspired by human cognition, and specifically, the human visual system. The original CNNs were based on the Neocognitron model, originally developed by Japanese computer scientist Kunihiko Fukushima in 1979 (Draelos, 2019). The original Neocognitron model is a hierarchical neural network with various layers. Each layer has the capability of extracting and identifying specific features of an image (Fukushima, Neocognitron: A hierarchical neural network capable of visual pattern recognition, 1987). The information learned by a layer is then transferred to the next layer, which performs a more detailed feature extraction task. An illustration of the Neocognitron model is shown in Figure 1.

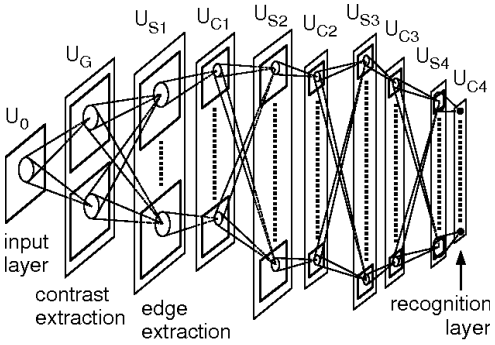

Figure 1. Architecture of the Neocognitron Model. Source: (Fukushima, Neocognitron for handwritten digit recognition, 2001)

Convolutional neural networks follow the same approach as the Neocognitron model, with some slight modifications. The first step towards the implementation of a CNN-based model is the data-preprocessing step. In this step, image pixels are converted to a computer-readable format (RGB numerical values). Once an array of numerical values is created, the input flows through multiple neural network layers, which perform computations to extract the image features and identify the object.

Modern CNNs mainly use three types of layers: convolutional, pooling, and fully-connected. Convolutional layers use filters for feature extraction. A filter, also known as a kernel, is an array of numerical values with a predetermined shape (see Figure 2). In a convolutional layer, the kernels are first superimposed on the image. Then, the dot

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

Figure 2. 3x3 kernels used for the extraction of vertical (left) and horizontal (right) features in images.

product between the filter values and the image RGB values is computed. The kernel (which has a smaller shape than the image) is slid across the image array to compute the dot product with the different sections of the image (Ke, et al., 2018). The resulting dot products are saved to a new array (output map), which serves as the input to the next layer.

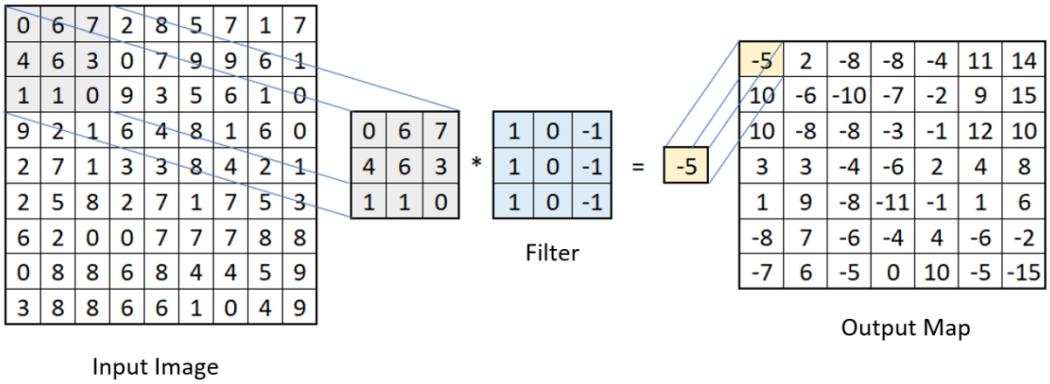

Figure 3. Convolutional layer with a 3x3 filter

The second type of layers used in CNNs is the pooling layer. The main goal of a pooling layer is to reduce the dimensionality of the output map (Teuwen & Moriakov, 2019), which is the output array of a convolutional layer (shown in the right side of Figure 3). Reducing the size of the image map is beneficial since it reduces the computational cost of training the neural network. For this reason, pooling layers are often added after every convolutional layer in a CNN. The output of the convolutional layer in Figure 3 is processed by a max pooling layer in Figure 4:

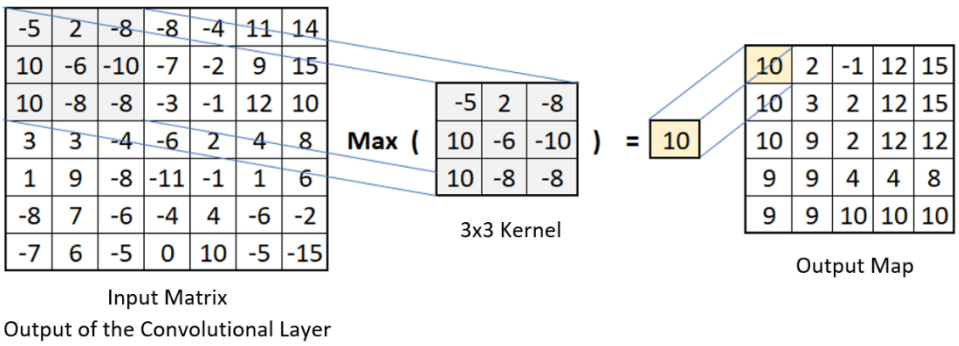

Figure 4. Max pooling layer with a 3x3 kernel and stride = 1.

The pooling layer follows a similar approach to the convolutional layer, where a kernel (an array) is superimposed on the image to extract features. In the case of the pooling layer, the kernel does not contain numerical values. Instead, the empty kernel is superimposed across the different sections of the image, and the maximum value of each section is saved to the output map. There is a second type of pooling, which computes the average value (rather than the maximum) and is known as the average pooling layer.

Convolutional neural networks also use fully-connected layers, known as dense layers. Fully-connected layers are the most common type of layers in deep learning (outside computer vision). A dense layer contains artificial neurons, which use parameters and non-linear functions to perform computations on the input data.

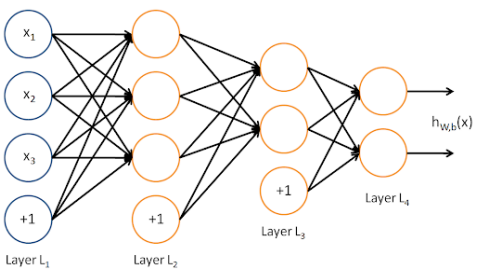

Figure 5. Neural network with three dense layers. L1 is considered an input layer. Source: (Stanford, 2018)

The main goal of training a neural network is to find the optimal set of parameters that reduces the error (or cost) of the model. On the other hand, the activation functions used

by a dense layer are defined by the user. Some examples include the sigmoid, SoftMax, and ReLu functions. Each dense layer in a neural network can be customized to have a different number of neurons or a specific activation function.

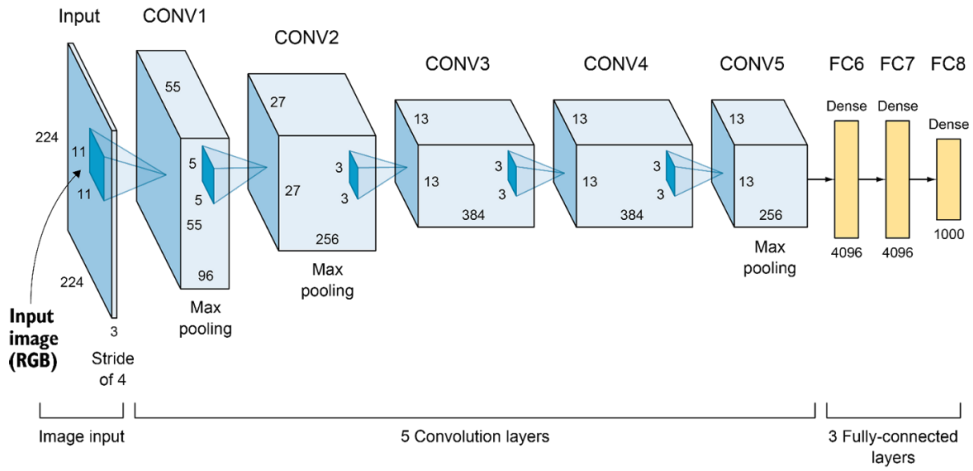

Figure 6. Architecture of a Convolutional Neural Network (AlexNet), with convolution, max pooling, and fully-connected layers. Source: (Elgendy, 2020)

Normally, fully-connected layers are the last step of a convolutional neural network (as shown in Figure 6). Dense layers use feature maps (obtained from the convolutional and pooling layers) to detect and classify objects in the image.

Convolutional neural networks can have many different shapes and parameters. Figure 6 shows one of the most popular CNN architectures (AlexNet), which was developed by Geoffrey Hinton and other researchers at Google Brain. In this project, an Inception Network was used for object detection and classification. The Inception Network model was originally developed by researchers at Google and is further explained in the methods section of this report.

The model for this project was created using TensorFlow and Google Colab, and can be found in the following GitHub repository:

https://github.com/nicolasgapa/Traffic-Modelling-using-Computer-Vision-CNNs-

You Only Look Once (YOLO)

Convolutional neural networks can have a variety of outputs. The most common application of CNNs is classification. In such model, the last layer contains a set of artificial neurons, each representing an object. For example, in a binary classification CNN which aims to identify whether the object in an image is a car or a boat, the CNN will have two neurons in its last layer, each corresponding to one of the objects. In a classification CNN, the output of each neuron is the probability of the input image being the object that the specific neuron represents, as shown in the figure below.

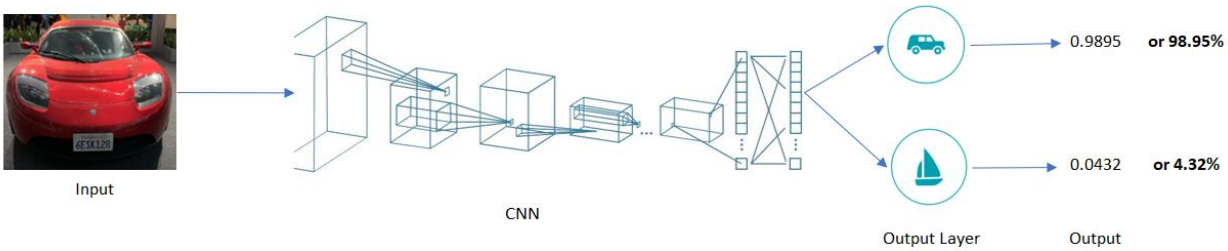

Figure 7. Classification task using a convolutional neural network.

Another type of convolutional neural network is the Mask Region-based CNN (R-CNN). Rather than returning a probability, the R-CNN model output is a matrix containing the probabilities, location, and size (bounding box) of the detected objects in the image (He, Gkioxari, Dollár, & Girshick, 2018). The capability of R-CNNs results more useful for industry applications since it can detect where the objects are inside the image, rather than just identifying the object type. The R-CNN performs does the object detection and segmentation tasks in two different steps.

A more recent model, known as You Only Look Once (YOLO), does both classification and object segmentation tasks in one step. While the fundamentals of YOLO and R-CNN are the same, the YOLO model proves to be faster (Gandhi, 2018). This project uses a pre-trained YOLO algorithm to classify objects and identify their location in the security camera images.

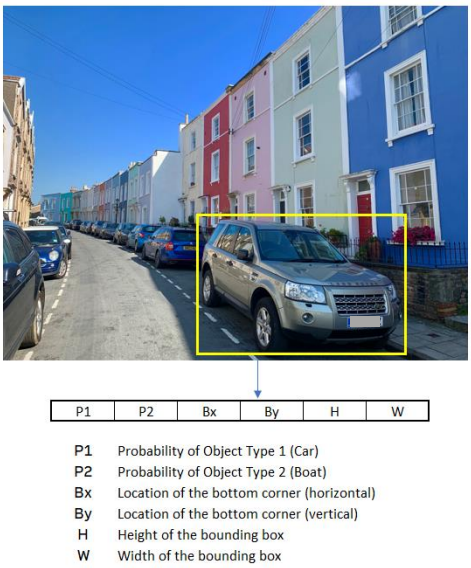

Figure 8. Output of a YOLO model, showing the class and location of the detected object in the image.

Transfer Learning

Transfer learning is a technique which consists of using previously trained neural networks to leverage the training of new models. In general, transfer learning is a way of utilizing previous knowledge obtained by old models to improve the performance and accuracy of a newly created model (Martin, 2019). Transfer learning applications are popular in computer vision and natural language processing, where models need to be trained using large data sets, and training may be computationally expensive and time consuming.

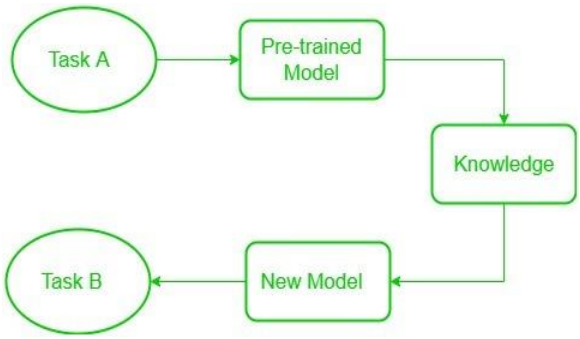

Figure 9. Outline of a Transfer Learning process. Source: (Chouhan, 2019)

A good example of why transfer learning is recommended in computer vision is object detection. In order to properly train a CNN to detect and classify objects, the model will need a large dataset with sample images to learn from. In most cases, each image contains millions of pixels, each comprising three numerical values (RGB). It is important to note that the accuracy of the model depends on the size of the dataset. Therefore, to improve the performance of a CNN model, more images are needed, resulting in a high demand of computational resources.

One way to reduce training time is to use a graphics processing unit (GPU), which allows concurrent (parallel) computations (OmniSci, 2020). GPUs can be used through online (cloud service) environments such as Google Colab and Jupyter Notebooks. However, even when using a GPU, training a computer vision model can prove to be a time-consuming task. In some cases, training a model can take weeks or months.

In this project, a pre-trained YOLO model was used for object detection. The model, called Darknet 53, is a convolutional neural network with 53 layers, trained with over one million images (Redmon & Farhadi, 2018), and was created by researchers at Cornell University.

Method and Implementation

Data Pre-Processing

The data used for traffic pattern modelling was collected using a security camera located in Bogota, Colombia. The recorded videos have a frequency of 12 FPS (frames per second) at standard HD resolution (1280x720 pixels). The OpenCV library in Python was used to process the video.

All video frames were reshaped to a size of 416x416 pixels, and the RGB values were normalized. Each pixel's RGB value falls within the range between 0 and 255. The normalization (or data scaling) process consists of dividing every RGB value by 255. The resulting array contains values within the range of 0 and 1. In general, training a neural network with normalized values is beneficial, since it reduces training time and leads to faster convergence (Stottner, 2019).

Figure 10. Security camera (left) and its view of the road (right). Location: Bogota, Colombia.

Classification

The object detection algorithm was developed using a pre-trained YOLO model known as DarkNet53. However, as part of this project, an Inception classification network model was also trained, using Python and TensorFlow Keras. The original implementation of the Inception network that was used for this project was developed by Faizan Shaikh, and is attached in the references of this report[1] (Shaikh, 2018).

Inception networks are composed of "inception modules", which are independent neural network modules containing convolution and pooling layers. The independent modules are concatenated to comprise the full architecture of the Inception network (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2015).

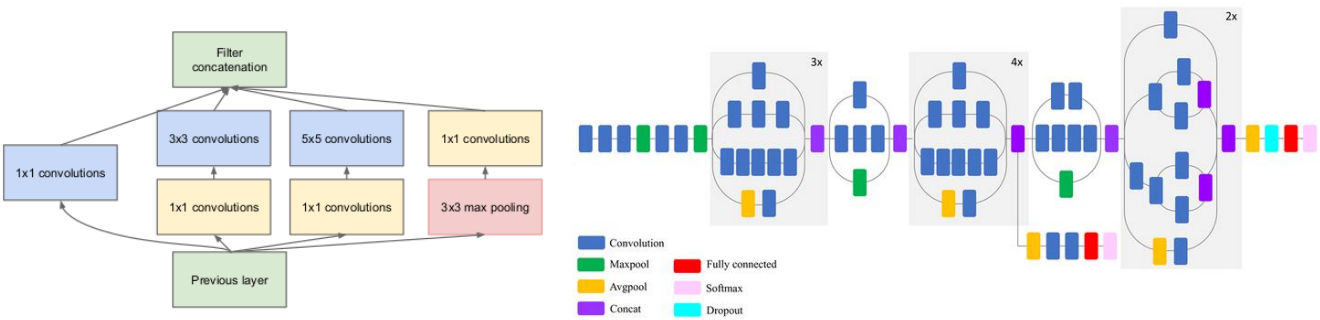

Figure 11. Inception module (left) and network (right). Sources: (Mahdianpari, Salehi, Rezaee, & Zhang, 2018) and (Shaikh, 2018)

[1] Inception network implementation: *https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/*

The architecture of an Inception network is different to those of traditional convolutional neural networks. However, the idea behind object detection and segmentation is the same, as well as the training process. The Inception network used for this project was created using Keras, a TensorFlow library (Shaikh, 2018), and can be found in the project's Github repository[2]. The classification network was trained using more than 10,000 images downloaded from the Common Objects in Context (COCO) dataset[3].

Transfer Learning

The DarkNet YOLO model can be imported from the OpenCV library in Python, while the pre-trained weights are available online[4]. OpenCV is a free-source library for computer vision applications useful for visual data processing (images and videos), and for the implementation of convolutional neural network models.

Since the DarkNet model only processes individual images, the camera recordings must be processed one video frame at a time. Once the pre-trained weights are loaded into the DarkNet model, each video frame is processed by the neural network. The output is a new image containing bounding boxes around the objects that were identified. The process is repeated for every video frame, using a *for loop* in Python. Moreover, a new video is created, containing the processed frames with the detected objects and their corresponding bounding boxes. The process is outlined in the following figure:

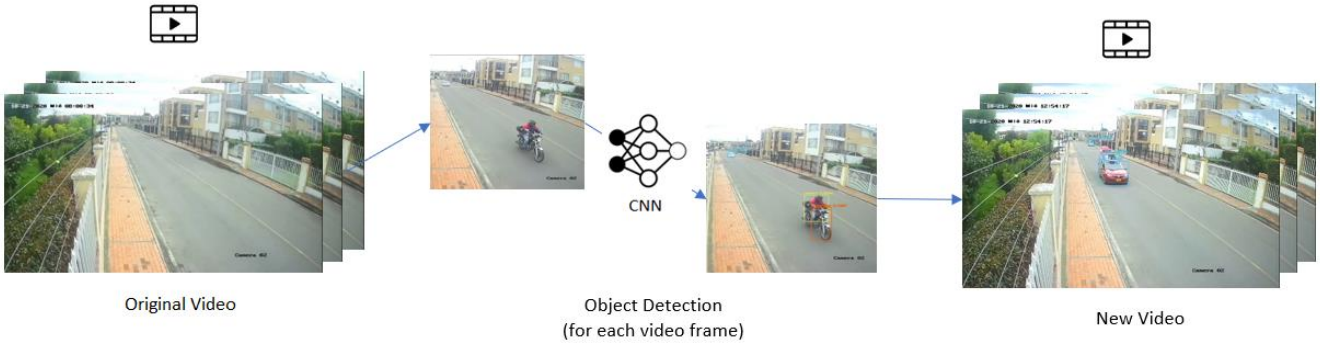

Figure 12. Video object detection process, using a YOLO convolutional neural network model.

Traffic Pattern Analysis

Traffic patterns can be inferred from the processed recordings by extracting the labels (object types) that were identified by the convolutional neural network. In order to reduce the computational expense of processing long recordings, the labels are extracted from the videos every second (rather than every video frame).

After the labels have been collected, they are saved to a Numpy file (.npy), which is processed by a different algorithm called *traffic_pattern_modelling.py* (refer to the project's Github repository). The python file loads the data extracted from the videos and generates plots for the identified object classes. The algorithm uses a moving average to determine the average traffic of each class during a specified period of time (window size).

For example, for a window size of 10 minutes, the algorithm would process 600 images (1 per second). Let us say that the neural network identifies cars in 400 out of 600 images. Moreover, out of

[2] Github repository: https://github.com/nicolasgapa/Traffic-Modelling-using-Computer-Vision-CNNs-
[3] COCO data set: *https://cocodataset.org/#download*
[4] Pre-trained weights of the DarkNet model: *https://pjreddie.com/darknet/yolo/*

those 400 images, the CNN detects 2 cars (per frame) in 300 images. The resulting moving average of the "car" object type during the 10-minute period is given by:

$$\frac{(300\ frames * 2\frac{cars}{frame}) + (100\ frames * 1\frac{car}{frame})}{600\ frames} = 1.17\ \frac{cars}{frame}$$

Results

Classification

The Inception network previously described was used to solve a classification task and can be found in the Github repository[5]. The neural network was trained using the GPU computing feature in Google Colab to improve performance and expedite the training process. The CNN was trained with a total of 13,071 images from the COCO dataset for five different classes: Cars (including trucks), motorcycles, bicycles, dogs, and horses. For the purpose of this project, the horse class was removed.

As previously mentioned, a classification neural network returns the probabilities of the image containing the object types. For this reason, the original Inception network model was modified so that its last layer would have 5 neurons, each returning the probability for one of the five pre-defined classes. The class with highest probability corresponds to the identified object, and the probability is defined as the neural network's confidence in its object detection. In order to test the model, a total of 728 images were set apart to compose the validation set, obtaining the following results.

Class	Validation Set Size	Accuracy	Confidence
Bicycle	150	48.67%	45.13%
Dog	150	86%	81.67%
Motorcycle	128	81.25%	74.89%
Truck	150	88.67%	83.03%
Total	**728**	**71.43%**	**66.31%**

Table 1. Validation set classification results – Inception Network

The Inception network performed well in three of the classes (*truck*, *motorcycle*, and *dog*), surpassing accuracy and confidence levels of 80%. However, the neural network's performance failed to properly classify images of *bicycles*, with an average accuracy and confidence level of 50%. One way to identify why the neural network has a limited performance (with respect to certain classes) is to determine how the model performs with respect to the training data.

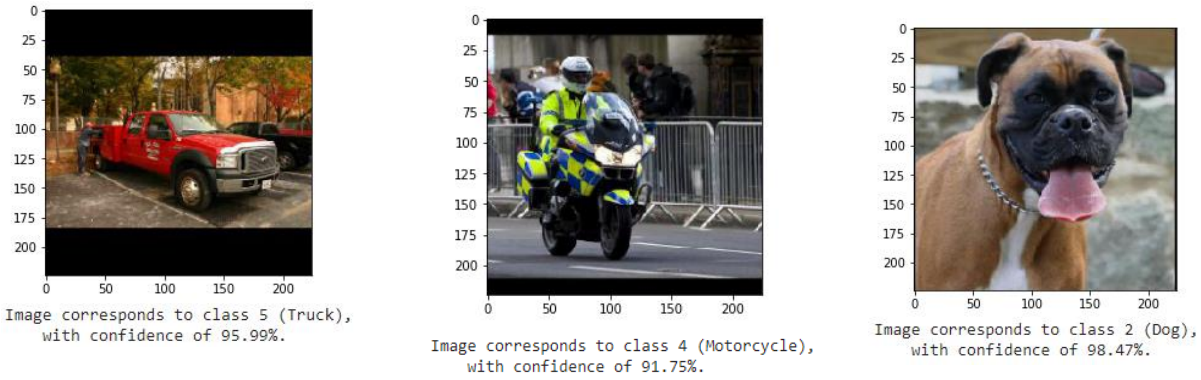

Figure 13. Images that were properly classified by the Inception network.

[5] Classification model – Link to Google Colab version: *https://github.com/nicolasgapa/Traffic-Modelling-using-Computer-Vision-CNNs-/blob/main/classification_model.ipynb*

Class	Training Set Size	Accuracy	Confidence
Bicycle	2740	83.5%	76.39%
Dog	2735	68%	51.94%
Motorcycle	2402	89.5%	80.22%
Truck	2467	81.25%	69.41%
Total	**13,071**	**84.17%**	**74.70%**

Table 2. Training set classification results – Inception Network

The results show that for the class that the Inception network failed to generalize (*bicycle*), the network had a high accuracy with respect to the training images. This is a sign of overfitting since the *bicycle* class' training accuracy (83.5%) was way above its validation accuracy (48.67%). Due to overfitting, many of the validation images were misclassified as a similar object. For example, many bicycle images were classified as motorcycles (as shown in the figure below).

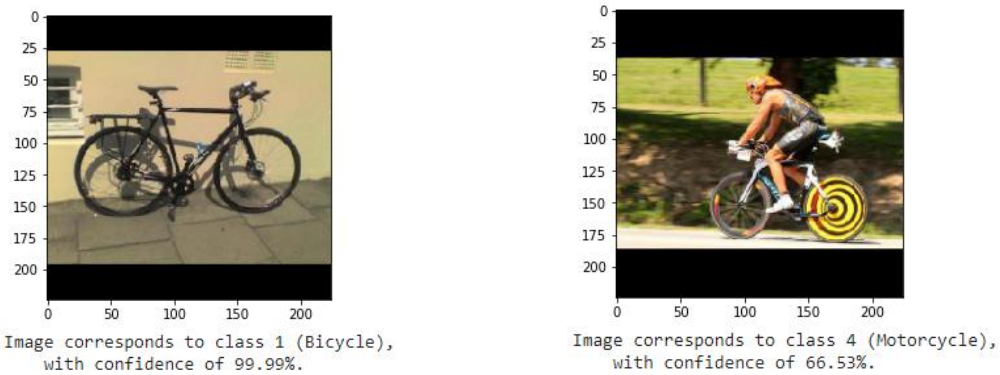

Figure 14. Image classification using the training (left) and validation (right) datasets.

There are multiple reasons why the Inception network may have failed to generalize well for all object types. The main reason could be the size of the training set. In general, computer vision systems need large datasets to be able to identify objects in images outside the training set. For instance, some models (such as DarkNet) are trained with more than a million images.

Another reason could be the limited training time and computational resources. Some computer vision models (using large training data sets) require powerful computers to be trained under a reasonable amount of time. For this project, the Inception network model was trained for 28 hours in total, using the GPU computing feature in Google Colab. Training the model for a longer period of time may produce better results. Once again, for the traffic modelling application, a pre-trained DarkNet model was chosen over the Inception network, allowing for better performance and the use of a YOLO object segmentation model.

YOLO for Video Object Detection

The neural network model for object detection, based on the DarkNet model, was implemented using Python and Google Colab, and can also be found in the project's Github repository[6]. Some sections of the code were based on an existing YOLO video object detection model, developed by Adrian Rosebrock, which is referenced below[7] (Rosebrock, 2018). The DarkNet model can identify and

[6] Video object detection model – Link to python model.
Link to Google Colab version.

[7] OpenCV model – Video detection model by Adrian Rosebrock:
https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/

locate more than 60 object classes, including cars, trucks, people, dogs, bicycles, and motorbikes. The following images are video frames extracted from security camara recordings and a video taken using an iPhone camara.

Figure 15. Object detection and segmentation using the DarkNet model.

The model was designed to be able to detect objects in individual images, as well as videos. The output of the model is the image (or video) with the respective bounding boxes, as well as a list of labels corresponding to the detected objects. Examples of processed image and video files are included in the Github repository.

Figure 16. Neural network model tested on a highway setting. Location: Chia, Colombia.

The model was capable of processing 2-3 frames per second. Since the camera records at 12 FPS, the model was modified to only process one (out of the 12 frames) per second. The modification would allow the model to process data in real-time. In some cases, Google Colab (with GPU computing) was capable of processing up to 5 frames per second, but this performance was not constant. The following table shows a summary of the processing times of the security camera recordings.

Date	Time processed	Number of frames	Google Colab's Processing Time
Day 1 (October 20)	7.5 hours	27,087	11,287s (3.13 hours)
Day 2 (October 21)	7.5 hours	27,158	10,052s (2.79 hours)
Day 3 (October 22)	7.5 hours	26,945	12,472s (3.46 hours)

Table 3. Data processing time using Google Colab

Traffic Pattern Analysis

The traffic pattern analysis was done using the video recordings of security cameras for a period of three days, October 20th-22nd. The plots for the following classes were generated: cars, people, motorbikes, and trucks. Once the identified labels were obtained from the CNN video processing step, the resulting arrays were imported and processed by the *traffic_pattern_modelling.py* algorithm[8].

The chosen window size of the moving average was 10 minutes (600 frames, each processed every second). The following plots show the traffic pattern results in the residential street where the camera is located. Due to computational and time constraints, only day images were processed, using Google Colab. As shown in the plots, some classes (such as cars and people) have a clear pattern, peaking around noon. For instance, at 12:00, there were an average of 2.5 cars per frame in the camera, indicating that the street had constant traffic at that time.

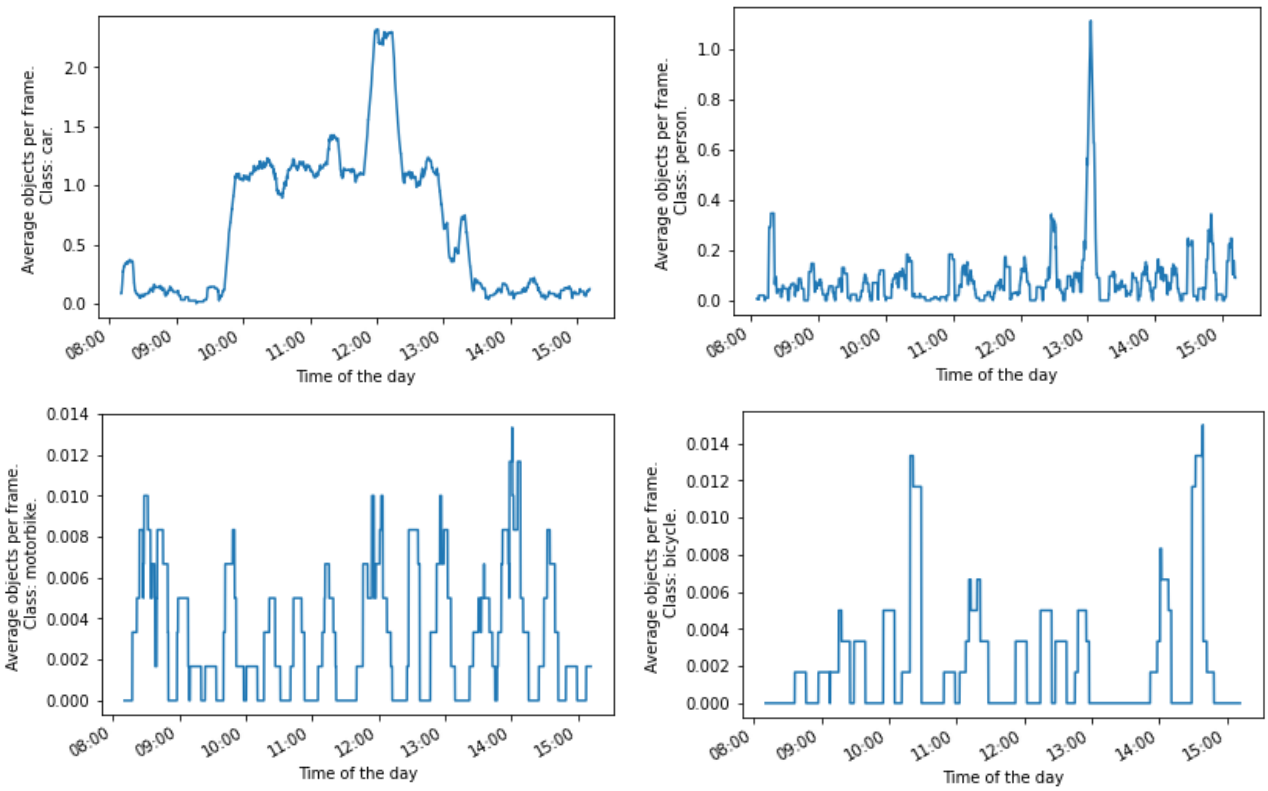

Figure 17. Traffic patterns in the street where the security camera is located.

Other classes (such as motorbikes, trucks, and bicycles) did not show a clear traffic pattern, having peaks and lows at different times of the day. Also, the plots show that the frequency at which bikes were identified was much lower than that of cars and people, as expected.

For reference, motorcycles peaked at an average of 0.013 motorbikes per frame (around 2 pm). This means that out of 600 frames (collected during a period of 10 minutes), motorcycles were detected in only 8 frames (0.013*600). In comparison, around 1 PM, the CNN detected on average 1 person per frame. In other words, people were identified in approximately 600 frames, in a period of 10 minutes.

It is worth noting that the method used in this project does **not** indicate the number of objects that moved along the street at any given time, but rather how many times each object type appeared in the

[8] Traffic pattern modelling algorithm – Link to Google Colab version.

video frames. For example, if a single person stands in front of the camera for an entire day, it might trick the algorithm to think that there is constant traffic of people in the street. Similarly, if a car is parked in front of the camera for a long period of time, such car would influence the traffic patterns. Further work could include improving the model so that it identifies each individual object only once, though it would increase the complexity of the model. Moreover, the model would perform better on a highway setting, were traffic stops are less frequent.

Following the same approach used to generate Figure 17, a plot showing the traffic pattern of *all* objects was generated (shown below). The plot shows that the traffic pattern of *all* objects closely aligns with the traffic pattern of *cars*. This indicates that most of the traffic on the road were cars, surpassing *people*, *bicycles*, and *motorcycles* by a large margin.

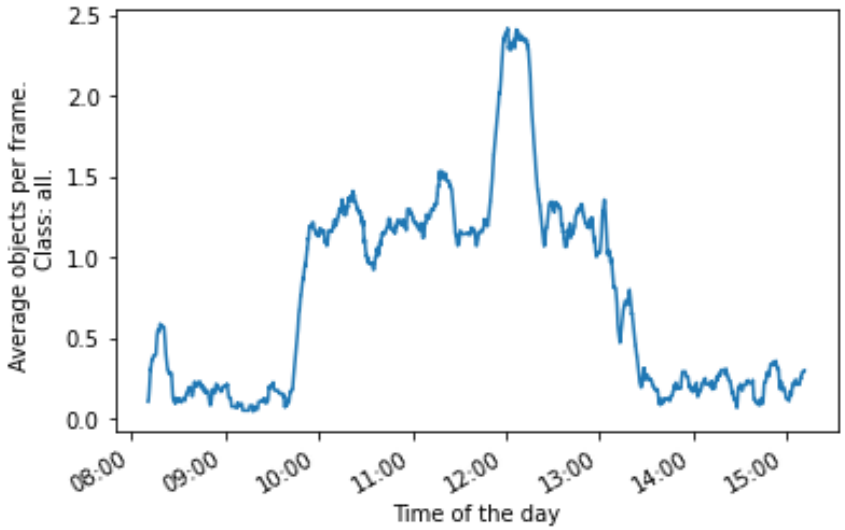

Figure 18. Traffic pattern of objects on the road (all classes: cars, people, bicycles, and motorcycles).

Conclusions and Further Work

Convolutional neural networks are a powerful tool to process and analyze digital visual data. The purpose of this project was to explore the use of deep learning and computer vision technologies, and to apply such knowledge to traffic data analysis. The project covered classification (Inception network), object detection, and object segmentation algorithms (YOLO).

First, the Inception neural network architecture was used to train a classification model. The network was capable of classifying images with an accuracy of 70%. The performance of the neural network was limited due to computational constraints. Moreover, the model performed better at identifying objects such as *cars* and *motorcycles*, while performing worse with the *bicycle* class. The accuracy of the neural network could be improved by adding more training data and training the model for a longer time. The Inception network was not used for the object segmentation task. Instead, a pre-trained DarkNet-53 neural network was used. The DarkNet model proved to be highly accurate, identifying objects of different shapes and sizes with high precision. Moreover, the model was capable of handling large video files, processing more than 3 images (video frames) per second in most cases.

The DarkNet model was used to analyze traffic data. The labels of the detected objects were saved to an array while the videos were being processed. Then, the collected data was used to plot and identify the traffic patterns of cars, motorcycles, bicycles, people, and trucks. In general, the model was successful at extracting data from the videos and generating traffic pattern plots. However, the model was tested in a residential street with limited traffic, and even though the results showed a clear trend

of traffic patterns in such street, the model should ideally be implemented on a highway setting. For instance, the model was tested using a short highway recording (1 minute), as shown in Figure 16, indicating that its performance is not constrained to the street size or to the number of objects present in the image.

Moreover, the video recordings used in this project were processed multiple days after they were recorded. Ideally, the model should process data in real-time, by continuously analyzing data from the security camera recordings. Finally, as new computer vision models are developed, the neural network model could be upgraded to improve its performance and speed, allowing more frames per second to be processed and expanding its ability to detect and locate objects in an image.

References

Chouhan, V. (2019, 11 25). *ML | Introduction to Transfer Learning*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/ml-introduction-to-transfer-learning/

Draelos, R. (2019, 04 13). *The History of Convolutional Neural Networks.* Retrieved from Glass Box Medicine: https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/

Elgendy, M. (2020). Chapter 5: Advanced CNN Architectures. In *Deep Learning for Computer Vision* . Manning Publications.

Fukushima, K. (1987, 09 15). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 119-130. Tokyo, Japan: NHK Science and Technical Research Laboratories Japan.

Fukushima, K. (2001). Neocognitron for handwritten digit recognition. *Neurocomputing*, 163.

Gandhi, R. (2018, 07 09). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms.* Retrieved from Towards Data Science: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018, 01 24). *Mask R-CNN.* Retrieved from Cornell University: https://arxiv.org/abs/1703.06870

Huang, T. S. (1996). *Computer Vision: Evolution and Promise.* Urbana, IL: University of Illinois at Urbana-Champaign.

Ke, Q., Liu, J., Bennamoun, M., An, S., Sohel, F., & Boussaid, F. (2018). Computer Vision for Human–Machine Interaction. In Q. Ke, J. Liu, M. Bennamoun, S. An, F. Sohel, & F. Boussaid, *Computer Vision for Assistive Healthcare* (pp. 127-145). Lecce, Italy: Academic Press.

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *Gradient-Based Learning Applied to Document Recognition.*

Mahdianpari, M., Salehi, B., Rezaee, F., & Zhang, Y. (2018). Very Deep Convolutional Neural Networks for Complex Land Cover Mapping Using Multispectral Remote Sensing Imagery. *Remote Sensing*, 1-21.

Martin, S. (2019, 02 07). *What Is Transfer Learning?* Retrieved from Nvidia: https://blogs.nvidia.com/blog/2019/02/07/what-is-transfer-learning/

OmniSci. (2020). *CPU vs GPU.* Retrieved from OmniSci.com: https://www.omnisci.com/technical-glossary/cpu-vs-gpu#:~:text=The%20main%20difference%20between%20CPU,resolution%20images%20and%20video%20concurrently.

Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement.* Seattle, Washington: University of Washington.

Rosebrock, A. (2018, 11 12). YOLO object detection with OpenCV.

Shaikh, F. (2018, 10 18). *Deep Learning in the Trenches: Understanding Inception Network from Scratch*. Retrieved from Analytics Vidhya:

https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/

Stanford. (2018). *Multi-Layer Neural Network*. Retrieved from deeplearning.stanford.edu: http://deeplearning.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/

Stottner, T. (2019, 05 16). *Why Data should be Normalized before Training a Neural Network*. Retrieved from Towards data science: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision.

Teuwen, J., & Moriakov, N. (2019). Convolutional neural networks. In J. Teuwen, & N. Moriakov, *Handbook of Medical Image Computing and Computer Assisted Intervention* (pp. 481-501). Amsterdam: Academic Press.

Thanki, R., & Borra, S. (2019). Application of Machine Learning Algorithms for Classification and Security of Diagnostic Images. In *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging* (pp. 273-292). Academic Press.

Yakura, Shinozaki, H. &., Nishimura, S. &., Oyama, R. &., & Sakuma, Y. &. (2018). Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism. *The 8th ACM Conference on Data and Application Security and Privacy*, (pp. 127-134). Tsukuba, Japan.