Vehicles Classification and Brand Recognition Using Convolution Network and Neural Networks

Jihene REZGUI ‡, Simon DAZÉ‡, Gaya MEHENNI‡
‡ Laboratoire Recherche Informatique Maisonneuve (LRIMa)
Montreal, Canada
jrezgui@cmaisonneuve.qc.ca

Abstract – **Autonomous vehicles are in full development and vehicles classification is a fundamental part of this new technology. To interact with other objects on the road, vehicles need to be able to identify what is surrounding them. In this paper, we develop a platform named VARC (Vehicle Algorithm for Recognition and Classification) integrating a fully connected neural network to classify different types of vehicles such as trucks. Moreover, VARC considers the detected type to identify the brand of the vehicle using a convolutional neural network. This allows getting valuable characteristics of the vehicle like its color, the number of passengers. By processing pictures taken from security cameras or from ones on vehicles, VARC may help cops identify stolen cars and autonomous vehicles improve their perception of their environment. VARC's neural network is trained on more than 7000 images of cars, trucks, and motorcycles. Results demonstrated the effectiveness of VARC in terms of generating valuable data while minimizing the needed resources.**

Keywords – Artificial Intelligence, Neural Network, Convolutional neural networks, Autonomous vehicles, image classification.

I. Introduction

With the massive attraction toward self-driving vehicles and using artificial intelligence in transports, vehicle recognition is now a major part of the future of transports. Vehicles need to be able to perceive and analyze what is surrounding them in an instant to be able to react properly. Doing so requires a capacity to discern different types of vehicle and classifying them. This challenge requires the development of an intelligent platform that would be implemented on vehicles. Different techniques are used to do image recognition and classification. The usage of convolutional neural networks (CNN) [1] is one of the most common and efficient ways to classify and identify images. Nevertheless, deep neural networks can be used too since CNNs need major computational power. In this paper, we combine both structures: (a) the deep neural network classifies vehicle and (b) the CNN focuses on brand recognition.

To be able to analyze an image, VARC needs to adjust the image to its parameters and then, it extracts the info inside it and convert those into matrices, which then serve as inputs for the network. In addition, it is important to note that neural networks and CNN need many images to train themselves to be able to categorize them. Then, it is able to receive brand new images as inputs which, as said before will be converted into matrices and analyzed by the system. **Our contributions i**n this paper, can be summarized as follows: (1) we introduce our platform named VARC which goal is to classify vehicles, which helps cops identify stolen cars and autonomous vehicles guide themselves; (2) we present the two networks used in VARC. A deep neural network and fully connected which VARC uses to classify vehicles and the convolutional network it uses to identify brands on vehicles. (3) We implement the two main algorithms of feedforward and backpropagation used in the neural network of VARC. (4) We present the structure of the convolutional network VARC uses to identify brand logos on vehicles. (5) We evaluate and discuss the results on vehicles classification, and we test the variation of parameters on our system and the effect they have on its accuracy.

II. Related work

Work on vehicle classification and recognition are, for the vast majority, using deep convolutional networks [1]. Some are trained to detect vehicles edges and relating those edges to a category, others try to identify the vehicle physiognomy on background and some try to identify features on a part of the vehicle image and map that information between categories.

Concretely, the usage of vehicle recognition is largely used in the conception of autonomous vehicles. Tesla is one of the companies which is a leader in this discipline. They created an autopilot system for their cars based on artificial intelligence for recognition. Their autopilot is powered by sensors and camera all around the car to be able to identify whatever surrounds the car. All those sensors serve as inputs for the neural net they are using. [2]

978-1-7281-4275-3/19/$31.00 ©2019 IEEE

The need to detect features and specific characteristics about an image for vehicle recognition and self-driving is what makes a convolutional network needed. NVIDIA was able to build a CNN based on only one input camera and they were able to make it control a car on road testing. Then they concluded that a CNN would be the perfect tool to develop autonomous driving. [3] However, the network trained by Nvidia 27 million connections and 250 000 parameters, making it pretty long to train and work with. That's why we limited the classification of VARC to a fully connected network to see if it could achieve a relevant efficacity a vehicle classification without carrying the weight of a big CNN.

III. VARC

A. *Inputs*

VARC takes images as inputs and they are converted (cropped) to fit our network's dimensions. We used several datasets to train our network. First, to calibrate the training and backpropagation we used the MNIST dataset to see if it was able to classify digits image and then, we proceeded with vehicles images from different images datasets. We trained it with pictures of cars, trucks, and motorcycles. We used different images to test the accuracy of the network and to see if it able to classify images it didn't use as training samples. For the convolutional network, we use images of cars which aren't cropped because we need to find details such as a brand logo on the picture.

B. *Platform analysis*

VARC allows the users to choose between two possibilities: they can use a pre-trained neural network or train their own neural network to identify types of vehicles.

1) Training : First, to train his own network, the user needs to select which dataset to use as the training set. He can make is selection by using a file chooser which will automatically go through the folder. Then, the user can choose how many EPOCHS he wants the network to do. The number of EPOCHS is the number of times the network will go through the dataset. While the network is training, the user can visually observe which images are being analyzed.

2) *Testing* : After the training phase, users can select an image to test the network with, they can even take one downloaded from the web. Then, the network will analyze it and return the percentage of classification for the vehicle type. If it is a car, the convolutional network will analyze the image too to try to find the brand and if it is able to do so, it will show the average characteristics of the brand's vehicles. That includes the average acceleration, the oil consumption, the number of passengers, etc.

3) Guide : VARC even offers its own window of help, detailing every aspect of it. The user can thereby understand it more easily.

C. *Neural Networks*

1) Structure

The neural network used in VARC is made of multiple layers: one input layer, two hidden layers and an output layer (see Fig. 1). Each layer has its own set of neurons, biases, and weights.

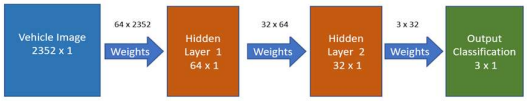

Fig.1. Structure of VARC's neural network

All hidden layers have access to a set of inputs, which is the output of the previous layer, and a set of outputs, which are the inputs of the next layer. The objective of a layer is to transform, by using a set of weights randomly generated connecting every input to every output, a set of inputs into outputs. This process, which is repeated for each layer of the network, is the ***feed forward propagation algorithm***, also called **FFPA**, which is explained in detail in [2]. In our case, the goal of the neural network is to identify a specific type of vehicles (cars, trucks, …). Each value of the input matrix corresponds to the RGB value of each pixel of a given image. We then process these values through each layer to get at the end of the neural network a set of numbers which can classify the image.

2) Feedforward algorithm

Each layer of the neural network transforms a set of given numbers into another set by using the ***feed forward propagation algorithm***. We decided to use the matrix version of the **FFPA** to simplify the notation and the programming of the algorithm. In the input layer, the weight matrix W is multiplied with the input matrix I which is a column matrix. Then, the bias matrix B is added to the product. We then apply the activation function to the sum to get the output matrix O. The output matrix of that layer becomes the input of the second layer and the process is repeated for each layer. Finally, the output of the last layer is supposed to classify the image.

$$O = \sigma(W \cdot I + B) \qquad \text{(Eq. 1)}$$

where O represents the column matrix of the outputs, W, the Weight matrix, I the Column matrix of the inputs, B the column matrix of biases and σ , the activation function.

VARC supports many activation functions: the sigmoid function, the hyperbolic tangent function, the linear function, the ReLU function, and the SoftMax function.

The sigmoid function can be defined as follows:

$$\sigma(x) = \frac{1}{1 - e^{-x}} \qquad \text{(Eq. 2)}$$

That function puts every input value into a range between 0 and 1.

The hyperbolic tangent function is similar to the sigmoid function since it compresses every value between a range, but the main difference is that that range is between -1 and 1. It can be defined as follows:

$$tanh(x) = \frac{2}{1 + e^{-2x}} + 1 \qquad \text{(Eq. 3)}$$

The third activation function supported by VARC is the linear function which can be described as follows:

$$f(x) = x \qquad \text{(Eq. 4)}$$

The ReLU activation function can be defined as follows:

$$f(x) = \begin{cases} x \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases} \qquad \text{(Eq.5)}$$

When the ReLU function or the linear function is used as an activation function in a neural network, the activation function of the last layer is set automatically to the SoftMax function to transform the outputs into probabilities. The SoftMax function's main goal is to set every output of the neural network in a range between 0 and 1 and to make them add up to 1. It can be defined as follows:

$$softmax(x)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \text{(Eq. 6)}$$

where z is a set of number and j is the index of the number in the set that we are applying the function to.

The feed forward algorithm, therefore, by assigning each output to a certain type of vehicle (car, truck, motorcycle), allows the neural network to guess the type.

3) Backpropagation algorithm

The ***backpropagation algorithm*** is an algorithm based on gradient descent; a mathematical tool used to find the minimum of a function. It is used to train regular neural networks. By feeding the neural network a certain input assigned with a certain label. That label corresponds to a certain output and by knowing the output that the neural network got with that input and knowing the actual output it is supposed to get, we can calculate the error of the neural network, also known as the cost function. Many cost functions exist, but the one we used can be defined as follows:

$$C = \frac{1}{N}\sum_{n=1}^{N}(\hat{y} - y)^2 \qquad \text{(Eq. 7)}$$

where $\hat{y}$ is the value obtained by the network, y is the expected value and N is the number of outputs of the network.

The goal of the backpropagation algorithm is to minimize that cost function by nudging each weight and each bias by a certain value with an algorithm called ***gradient descent*** **(GDA)**. That algorithm is based on finding the gradient vector, a vector using the partial derivatives of a multivariable function that indicates the direction of the maximum of that same function. By taking the opposite direction of that vector, the **GDA** is able to find the minimum. Depending on the size of the neural networks and the number of neurons in each layer, the number of dimensions of that function can go from 1 to infinity. Finding the absolute minimum of a function like this is almost impossible, but we can find a local minimum which gives a high success rate.

That algorithm, on the other hand, only gives the direction of a minimum at a local point. Thus, after each step, we need to recalculate the gradient vector and take another step, because the vector saved doesn't indicate the direction of the minimum at the new point. That process is repeated after each step. The size of each step is called the learning rate.

To train our network, we first calculate the error of the final layer using the cost function and the partial derivative of the activation function evaluated at the output. We then make the changes in the weights and biases matrices in the output layer. Once we got the error of the last layer, we evaluate the error of the last hidden layer using the error of the output layer. We then change the weights and biases of that hidden layer to minimize the cost function a little bit more.

If the neuron is an output neuron, the error can be calculated with this formula:

$$\delta_j^L = (\hat{y} - y) \cdot f'(\hat{y}) \qquad \text{(Eq. 8)}$$

where δ_j^L is the error of the j[th] neuron in the last layer, $\hat{y}$ is the output value, y is the expected value and $f'(\hat{y})$ is the derivative of the activation function evaluated at the output value

If the neuron is in a hidden layer, the error of each neuron can be calculated with this formula:

$$\delta_j^l = (\sum_{i=0}^{l+1} w_{j,i}^{l+1}\delta_j^{l+1}) \cdot f'(o_j) \qquad \text{(Eq. 9)}$$

where δ_j^l is the error of the j[th] neuron at layer l,

$w_{j,i}^{l+1}$ is the weight in the layer l+1 connecting the j[th] neuron in the layer l+1 to the i[th] neuron in the layer l,

δ_j^{l+1} is the error of the j[th] neuron at layer l+1 and $f'(o_j)$ is the derivative of the activation function evaluated at the output of the j[th] neuron.

The matrix form of these equations is much simpler to use and can be defined as follows [4]:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^L = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad \text{(BP4)}$$

BP1, as shown above, is the backpropagation of the output layer, BP2 is the error for each hidden layer and BP3 and BP4 show the derivative respect to each bias and each weight.

That process is repeated until the input layer which doesn't have any weights and biases to update. This algorithm is called the ***backpropagation algorithm*** because it goes by the neural network backward.

4) Dataset

Even though the **BPA** and the **FFPA** are the core of neural networks, they need something else to get a general idea of what a vehicle is: data. To make the training more efficient, neural networks need to see the data in random order. The reason is simple: if we give it thousands of images of cars and then thousands of images of trucks, it will first adjust to recognize a car and then a truck, but never both at the same time. Therefore, the dataset needs to be set in a random order for the network to get a general idea of what a car and a truck is.

D. ***Convolutional neural network***

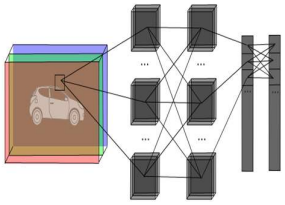

Fig. 2. Convolutional network structure and color channels

Convolutional neural networks, also called CNNs, are enhanced versions of neural networks for image recognition. They are able to recognize patterns and merge these patterns to guess the shape of an image and classify it. They are composed of various kinds of layers (see Fig. 2) like the convolutional layer, the max pooling layer, the vectorization layer, and the fully connected layer and many more. Those used to create VARC will be defined below.

1) Convolutional layer

Convolutional layers are the most important layers in CNN. Their goal is to detect edges inside an image by using filters (see Fig.4) The filters slide through the image (converted as a matrix) and returns a new smaller matrix where edges and patterns are highlighted. This sliding consists in the operation of this layer: the convolution product, which is based on the Hadamard product. The outputs of those layers consist of smaller matrices. The size of the outputs depends on the filter size and can be defined as follows:

$$O = I - F + 1 \quad \text{(Eq.10)}$$

Where O is the output matrix size, I is the input matrix size and F is the filter size.

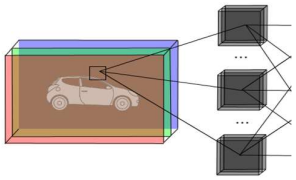

Fig. 3. Convolutional layer

2) Max pooling layer

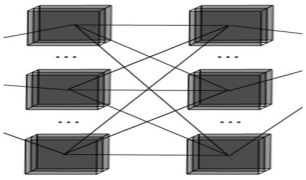

Fig. 4. Max pooling layer

The goal of the max pooling layer is to extract the most important information about a matrix. To do so, a filter slide through the matrix and extract the maximum value of each sub-matrices generated, then it stocks those values in a new smaller matrix which is the output of those layers. Max pooling layers are always after convolution layers to extract the most meaningful information highlighted by the convolutional layers. There is usually one or two convolutional layers followed by a max pooling in a CNN.

It is important to note that there are other ways to pool information out of a matrix in a CNN like by pooling the average value of the submatrices.

3) Vectorisation layer

The goal of a vectorization layer is simply to transform a 3D set of inputs into 1D. Indeed, since the output of a convolutional layer or a pooling layer is always an array of matrices and that at the end, we must have a single array, we need a layer to transform 3D parameters into 1D (see Fig. 5)

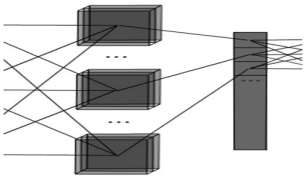

Fig. 5. Vectorisation layer flattening 3D inputs

The vectorization layer is always the second last layer of the CNN because its objective is to transform the input of a convolutional layer or a pooling layer into a set of inputs a fully connected layer can take.

4) Fully connected layer

A fully connected layer is the same layer as one in a regular neural network (see Fig. 6). It has weights and biases and uses the same two algorithms to train (**FFPA** and **BPA**)

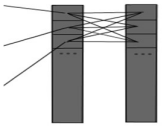

Fig. 6. Fully connected layer

5) Feed Forward

The ***feed forward propagation algorithm*** **(FFPA)** for convolutional neural networks is a more complex algorithm than the one for regular neural networks because the operation to transform inputs into outputs is different in every layer. For each layer, we apply its own operation. We then transmit the output of that layer to the next and repeat the process until we get to the final output of the network. A convolutional neural network always ends with a ***vectorization layer*** and a ***fully connected layer*** to transform the 3D input into an array of numbers and classify an image from the given one-dimensional input.

6) Backpropagation

The ***backpropagation algorithm*** **(BPA)** in a convolutional neural network is also similar to the one for neural networks with a few differences. Indeed, since not every input is connected to every output, the partial derivatives differ, but the idea remains the same: with the cost function, get the error of a layer, use that error to change the parameters like the weights and biases and propagate that error in the previous layers. Considering each layer is different, the equations for the errors of each layer is different. If we use the same cost function, the error of each layer can be defined as follows:

The error of a fully connected layer (see section D.4) :

$$\Delta\delta = W^T \times \Delta\hat{y} \qquad \text{(Eq. 11)}$$
$$\Delta W = \Delta\hat{y} \times I^T$$
$$\Delta B = \Delta\hat{y}$$
$$\Delta\hat{y}(i) = (\hat{y}(i) - y(i)) \cdot f'(\hat{y}(i))$$

where, $\Delta\delta$ is the error of the layer, W is the weight matrix, $\hat{y}$ is the output of the network, y the expected value, I the inputs matrix, ΔW is the matrix with the variation of each weight and ΔB is the matrix with the variation of each bias.

Updated weights and biases of a fully connected layer (Section D.4)

$$W = W - \eta \cdot \Delta W \qquad \text{(Eq .12)}$$
$$B = B - \eta \cdot \Delta B$$

where, W is the weight matrix, ΔW is the matrix with the variation of each weight, ΔB is the matrix with the variation of each bias and η is the learning rate.

The error of a vectorization layer (Section D.3) :

Since there are no variables to update in a vectorization layer, we just need to transform the flattened errors back into an array of matrices.

The error of a convolutional layer (Section D.1):

$$\Delta\delta_i^{l-1} = \sum_{f=1}^{F} \Delta C_{f,\sigma}^l * k_{f,i}^l \qquad \text{(Eq. 13)}$$
$$\Delta C_{f,\sigma}^l(i,j) = \Delta C_f^l(i,j) \cdot f'(O_f^l)$$

where $\Delta\delta_i^l$ is the error of input i of the layer l, ΔC_f^l the error of the filter f in the layer l, $f'(x)$ the derivative of the activation function and O_f^l the output value of filter f of layer l.

* *Denotes a full convolutional operation, not a valid convolutional operation like the one used in the* ***FFPA****. The main difference is that in a valid convolutional operation, we add padding before applying the operation. In that case, the padding is the size of the filter - 1.*

Updated weights and biases of a convolutional layer (see section D.1)

$$\Delta k_{i,f}^l(u,v) = \sum_{p=l}^{t}\sum_{q=l}^{t} \Delta\delta_f^{l+l} \cdot f'(O_f^l) \cdot I_i^l(p-u, q-v) \qquad \text{(Eq. 14)}$$

where $\Delta k_{i,f}^l(u,v)$is the variation of the weight indexed at u and v of the filter f at layer l for the input i, t is the size of the output matrix for the input i and filter (we consider that it is a square matrix, $\Delta\delta_f^{l+l}$is the error of the output f in the next layer (there are as many filters as outputs), $f'(x)$ is the derivative of the activation function, O_f^l is the output value of filter f of layer l and $I_i^l(p-u, q-v)$is the input value of the i^{th} matrix at layer l of index $(p - u)$ and $(q - v)$.

IV. Platform Results

For our testing, we used two hidden layers and the following configuration of neurons (see Fig.1 for more details):

2352, 64, 32, 3

Here, each number represents the number of neurons of the layer. In order from the input to the output.

First, we tested the neural network with vehicles classification, and we varied the learning rate and the number of epochs to highlight the best combination and how

they influence each other. We used the MIO-TCD dataset to train our networks.

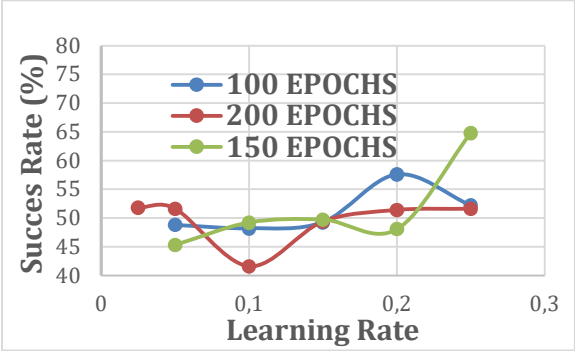

Fig. 7. Success rate depending on the learning rate for a training with 3000 images/EPOCH to classify vehicles between 3 categories

Thereby testing with several learning rates with achieved a success of 64.8% with 150 EPOCHS and 0.25 of learning rate. Also, we denoted that the best learning rate was between 0.15 and 0.25 for our neural network. However, the training phase was a bit long, so we tried to reduce the number of images per EPOCHS and we retested with the same parameters.

There we were able to achieve practically the same accuracy by reducing the training time and the number of images used to train the network. Also, we confirmed that the most efficient parameters for our network (we small numbers of EPOCHS) is to use 150 EPOCHS and learning rate of 0.25.

With those results, we demonstrated that a high learning rate (0.25) coupled with a relatively small number of EPOCHS can have a significant success rate. In fact, in both Fig. 7 and Fig. 8 we achieved more than 60 % accuracy with 150 EPOCHS and 0.25 learning rate. Also, with 1000 images per EPOCHS VARC achieved 63.93 %, it's 0.9 less than with 3000 images per EPOCHS. By doing so, we found that a simple neural network can achieve significant results in vehicle recognition despite not having convolution layers and millions of parameters.

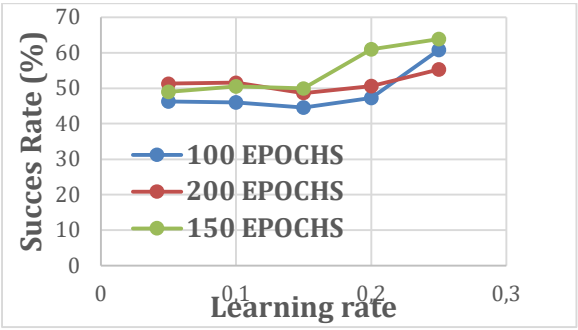

Fig. 8. Success rate depending on the learning rate for a training with 1000 images/EPOCH

Acknowledgment

We would want to thank Simon Vezina and Caroline Houle, professors at *College de Maisonneuve*, for their help regarding this project in the implementation of the neural network and convolutional neural network.

References

[1] D.T, MUNROE and G.M, MADDEN. "Multi-Class and Single-Class Classification Approaches to Vehicle Model Recognition from Images." AICS, pages 93-102. 2005

[2] TESLA. "Autopilot." 2019. https://www.tesla.com/autopilot [last visit and update 30/04/2019]

[3] M. Bojarski. et al. "End to End Learning for Self-Driving Cars." airXiv.1604.07316. (April 25, 2016).

[4] M. Nielsen. "Neural Network and deep Learning. Chapter 2: How the backpropagation algorithm works.", 2015.

[5] Medium. "Understanding of Convolutional Neural Network (CNN) - Deep Learning." https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 [last visit 20/04/2019]

[6] Z. Zhang. "Derivation of Backpropagation in Convolutional Neural Network (CNN)." 2016.

[7] A. Ng. DeepLearning.ai: "Convolutional Neural Networks: course 4 of the deep learning specialisation." YouTube. (2017).

[8] L. Zhuo, et al. "Vehicle classification for large-scale traffic surveillance videos using Convolutional Neural Networks."Machine Vision and Application. (2017).

[9] ShutterStock.INC.Carimage. https://image.shutterstock.com/image-vector/black-white-car-260nw-226838140.jpg

[10] S. Saha. "A Comprehensive Guide to Convolutional Neural Networks- The EL15 Way." 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [last visit 18/04/2019]

[11] G. Gwardyz. "Convolutional Neural Networks backpropagation: from intuition to derivation." WordPress. 2018.

[12] Z. Luo, et al. "MIO-TCD: A new benchmark dataset for vehicle classification and localization in press at IEEE Transactions on Image Processing".2018.

[13] Jefkin. "Backpropagation in Convolutional Neural Networks." Deep Grid. 2016.

[14] Stanford University. "CS231n: Convolutional Neural Networks for Visual Recognition." 2018.

[15] M. Agarwal. "Backpropagation in Convolutional Neural Networks - Intuition and Code."Medium. 2017.