Car image recognition using convolutional neural network with efficient met architecture

Serhiy Balovsyak[1,†], Olga Kroitor[1,†], Khrystyna Odaiska[1,†], Abdel-Badeeh M. Salem[2,†] and Serhii Stets[1,*,†]

[1] *Yuriy Fedkovych Chernivtsi National University, Kotsiubynsky 2, 58012, Chernivtsi, Ukraine*
[2] *Ain Shams University, El-Khalyfa El-Mamoun Street Abbasya, Cairo, Egypt*

Abstract

A convolutional neural network for car image recognition has been developed. The neural network model is implemented with the EfficientNet architecture. The software implementation of the neural network was written in Python using the Keras library. The neural network model was trained on the basis of the publicly available Vehicle Detection Image Set dataset, which made it possible to compare the accuracy of the developed model with analogues. The structure of the developed model with the basic architecture of EfficientNet was improved, which increased the accuracy of model training. The improvement consisted of adjusting the network parameters, unfreezing some of its layers, and creating stop conditions to avoid overtraining. These additional measures helped to improve the model's accuracy. The images used to train the model were pre-processed by scaling and randomly rotating them (augmentation). Due to the model improvement and data preprocessing, a high model training accuracy (99.98%) was obtained, which exceeds the training accuracy for the best analog model (99.63%) on the used dataset. The developed convolutional neural network can be used for car image recognition and localization.

Keywords

Artificial Neural Network, Convolutional Neural Network, Efficientnet, Car Image Recognition, Python

1. Introduction

In today's world, there is a need to develop Artificial Neural Networks (ANN) designed to recognize digital images. In particular, in practice, the task of recognizing car images often arises. This task is quite effectively solved by Convolutional Neural Networks (CNN) [1-3], which use deep learning technologies. CNN are particularly effective in image recognition, as they take into account the geometry of images, the peculiarities of human visual perception, and use multilayer signal processing for a large number of examples. The name

IntelITSIS'2024: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 28, 2024, Khmelnytskyi, Ukraine
* Corresponding author.
† These authors contributed equally.
s.balovsyak@chnu.edu.ua (S. Balovsyak); o.kroitor@chnu.edu.ua (O. Kroitor); k.odaiska@chnu.edu.ua (Kh. Odaiska); abmsalem@yahoo.com (Abdel-Badeeh M. Salem); stets.serhii@chnu.edu.ua (S. Stets)
0000-0002-3253-9006 (S. Balovsyak); 0000-0003-4541-3805 (O. Kroitor); 0000-0002-3167-1195 (Kh. Odaiska); 0000-0003-0268-6539 (Abdel-Badeeh M. Salem); 0009-0007-0231-9970 (S. Stets)

of CNN is explained by the convolutional operation used in them, which is performed when processing two-dimensional signals.

However, when recognizing car images using existing convolutional neural network architectures, difficulties arise due to the low training speed of the CNN and limited recognition accuracy. The significant training time of the CNN is explained by the large number of layers of the neural network and the large size of the training sample. Insufficient accuracy of object recognition is explained, in some cases, by the incompleteness of the training set. One way to improve the accuracy of CNN is to increase the number of neural network parameters, but this can lead to overtraining and a decrease in recognition accuracy for images that do not belong to the training set. In general, the recognition accuracy decreases due to the mismatch between the architecture and training parameters of the CNN and the features of the images of the objects to be recognized.

Therefore, an important task is to develop a convolutional neural network that would ensure high accuracy of car image recognition with acceptable CNN training time. This task is realized by selecting the appropriate CNN architecture, pre-processing the images of the training and control samples, and adapting the structure and training parameters of the CNN to the features of car images.

2. Related Works

The architecture of the CNN is a determining factor that affects the quality of the network. Therefore, let's consider the main modern CNN architectures that can be used for car image recognition.

ResNet (Residual Networks) architectures, especially its deep variants ResNet-50 or ResNet-101, are widely used and have proven to be effective in image classification tasks, as shown in [3]. They utilize residual block connectivity, which helps to reduce the problem of vanishing gradients and improves the training of deep networks. Typical ResNet models are implemented with double or triple layer transmissions, with nonlinear activation functions (e.g., ReLU), or with batch normalization in the middle. The disadvantage of deep ResNet models is the long training time due to problems with gradients. In addition, an increase in the number of layers leads to an increase in the number of parameters, which requires more computing resources.

DenseNet architectures [4, 5] are known for their dense connectivity, which facilitates feature reuse and facilitates the flow of gradients during training. DenseNet models have shown high performance in image classification tasks and can be effective for identifying cars in images. However, the dense connections of such CNN can lead to an increase in the number of parameters, which can make model training more difficult.

The Xception architecture [6] uses deep separate convolutional units, which reduces the number of parameters (with a fairly high performance). Such CNN are successful in various image classification tasks. However, neural network models with the Xception architecture may require long training due to their depth and complexity of architecture. In addition, when deep convolutional units are used, the problem of learning with vanishing gradients may arise. Compared to other architectures, such as ResNet or DenseNet, Xception may be less efficient in terms of recognition accuracy and training resource requirements.

The EfficientNet architecture [7] differs from other architectures in that it is balanced, as it achieves a trade-off between accuracy and training time. EfficientNet uses scaling of network width, depth, and resolution to achieve an optimal balance between accuracy and computational efficiency. This ensures high accuracy of image classification with minimal resource utilization. In addition, EfficientNet's well-balanced architecture uses deep connections between layers and optimized convolutional units to help avoid problems with vanishing gradients and ensure stable training even for deep models. The EfficientNet architecture can be successfully used for a wide range of tasks from image classification to object detection. For this reason, this architecture is often used in the computer vision [8-10].

Therefore, taking into account the advantages and disadvantages of the considered CNN architectures, the EfficientNet architecture (its basic variant EfficientNetB0) was chosen for the software implementation of the car image recognition system [11- 13].

Real images of cars are obtained from different angles and at different distances from the video camera [14], and are characterized by uneven lighting, noise, and foreign objects. Therefore, in order to recognize car images with high accuracy, it is necessary to perform image preprocessing before training the CNN [15-17]: scaling to a given size, obtaining images from different angles by rotating them.

3. Convolutional Neural Network Model

For car image recognition, CNN model was developed with the EfficientNet architecture, which is characterized by an optimal balance between recognition accuracy and the number of floating-point operations (FLOPS), making it very effective for various computer vision tasks [18-20]. The structure of EfficientNet (Table 1) consists of a number of stages, which include various convolution operations and mobile feedback units MBConv. Each stage has its own input parameters, such as image dimensions (H, W), number of channels (C), and number of layers (L).

Table 1

EfficientNet-B0 baseline network; each row describes a stage i with L_i layers, with input resolution (H_i, W_i) and output channels C_i [7].

Stage i	Operator F_i	Resolution $H_i \times W_i$	#Channels C_i	#Layer L_i
1	Conv3×3	224 × 224	32	1
2	MBConv1, k3×3	112 × 112	16	1
3	MBConv6, k3×3	112 × 112	24	2
4	MBConv6, k5×5	56 × 56	40	2
5	MBConv6, k3×3	28 × 28	80	3
6	MBConv6, k5×5	14 × 14	112	3
7	MBConv6, k5×5	14 × 14	192	4
8	MBConv6, k3×3	7 × 7	320	1
9	Conv1×1 & Pooling & FC	7 × 7	1280	1

MBConv is the primary block used in the EfficientNet architecture. It is based on the mobile inverted bottleneck convolution and squeeze-and-excitation optimization. This block contributes to optimizing the number of parameters and improving the convolutional operation process. Additionally, EfficientNet differs from other architectures in that it has optimally tuned scaling coefficients (α, β, γ), allowing for network size increase through compound scaling [7]. This enables larger models with high accuracy while maintaining an optimal balance between the number of parameters and computational complexity.

Starting from the baseline EfficientNet-B0 model, the compound scaling method is applied to scale it up in two steps:

STEP 1: Initially, φ = 1 is fixed, assuming twice as many resources available, and a small grid search is conducted for α, β, γ values. φ represents the compound scaling factor, which is used to scale the network width, depth, and resolution simultaneously to achieve optimal performance.

STEP 2: Then, α, β, γ are fixed as constants, and the baseline network is scaled up with different φ values.

The EfficientNet model easily adapts to different image sizes and accuracy requirements through compound scaling. This allows optimizing the model for specific task conditions, such as varying input image sizes or performance requirements.

4. Software Implementation of Convolutional Neural Network

4.1. Selection of Training and Control Datasets

The developed CNN with the EfficientNet architecture was trained on the basis of the Vehicle Detection Image Set, which contains 17760 color images, of which 8792 images contain cars (Fig. 1) and 8968 do not (Fig. 2). The dataset was split into training and test sets with a ratio of 0.7, respectively, to ensure the accuracy of the model evaluation. The division of images was made random. The images in the dataset have a size of 64 × 64 pixels. The dataset is available on the kaggle.com website [21], which makes it possible to compare the performance of the proposed model with existing works. Before using the dataset, it is necessary to perform processing and preparation steps to work with the selected neural network architecture.

Figure 1: Fragment of the image dataset with cars (12 images out of 8792).

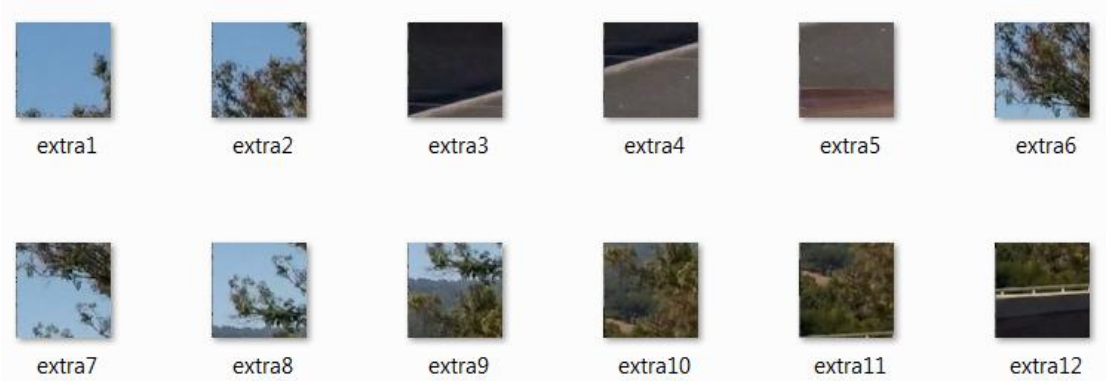

Figure 2: Fragment of the image dataset without cars (12 images out of 8968).

4.2. Preliminary Image Processing

Before training the CNN, images are loaded from the catalogs. The program accepts paths to directories containing images with and without cars. After that, the images are loaded using OpenCV for the processing stage. Each loaded image is scaled using the reshaped_image method. This method converts the image to a size of 100 × 100 pixels. If the original image is not square, then additional pixels are added to it or unnecessary ones are removed (to maintain proportions). An algorithm has been developed that transforms rectangular images into square ones. To reduce distortion in scaled images, the color of the added pixels is calculated as the average value of the nearest half of the image (Fig. 3).

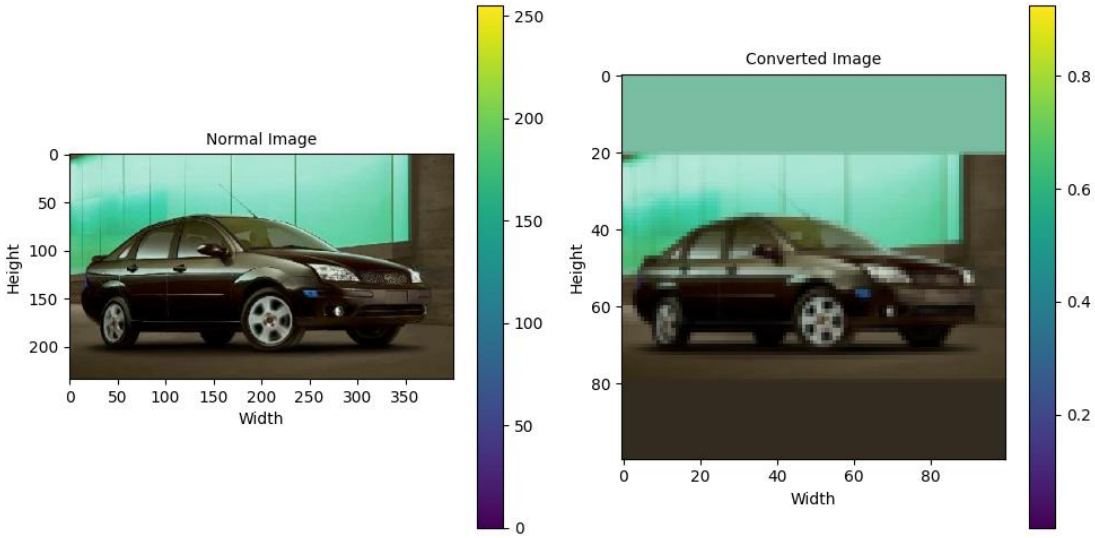

Figure 3: An example of how the algorithm works to transform a rectangular image into a square one.

This scaling allows to process images of any proportions with by CNN. After scaling, the color channel values of all pixels in the image are normalized in the range from 0 to 1. This improves the convergence of the neural network training. These steps ensure that the images are prepared for use in the neural network for classification.

4.3. Data Augmentation

One approach to data augmentation is to apply random rotation of images with a small tilt angle. This approach aims to increase the diversity of the data and improve the overall ability of the model to generalize new images. In the developed program, each input image from the dataset is rotated twice: first to the right and then to the left by a random angle (for example, between -10 and 10 degrees). Thus, each original image is complemented by two rotated images. This allows the CNN model to learn to recognize objects in images from different viewing angles, making it more robust to changes in the position or angle of objects (e.g., cars).

An important advantage of data augmentation is that it increases the stability of the model. The CNN becomes more adaptable to different shooting conditions (viewing angles, lighting, etc.). This method also helps to avoid overtraining, as the model is trained on more diverse data, which makes it capable of generalizing new images. Increasing data diversity also helps to make the model less sensitive to noise and other artifacts in the data. A visualization of image augmentation is shown in Figure 4.

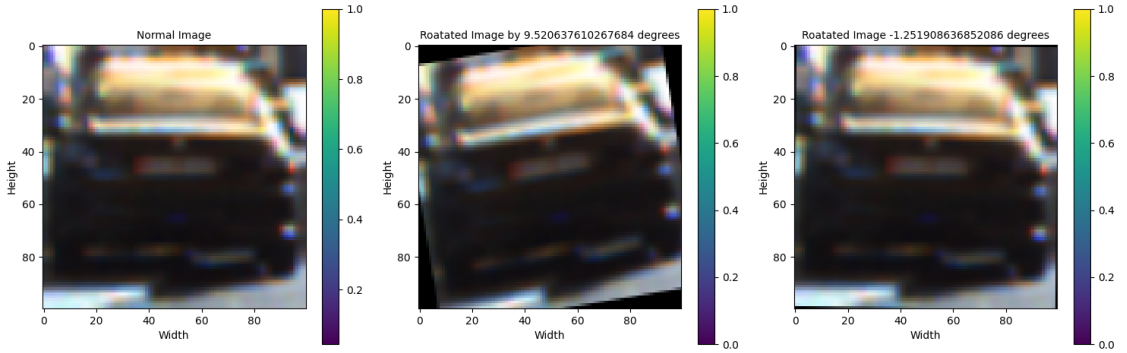

Figure 4: An example of data augmentation using image rotation.

However, it should be noted that data augmentation leads to an increase in the size of the dataset and an increase in the computational cost of model training. After augmentation, the dataset is increased to 53,280 images, of which 42624 are used for training and 10656 for validation.

4.4. Software Implementation of a CNN with EfficientNet Architecture

In this work, the CNN model with the EfficientNetB0 architecture was implemented using the Python programming language and the Keras library (Table 1). Also, layers for global average pooling were added to the model, followed by two fully connected layers for classification, and the last layer used the softmax activation function. The program code to implement the model based on EfficientNetB0 uses the Keras framework, which provides a user-friendly interface for building and training neural networks. The first step is to load the pre-trained EfficientNetB0 model without the top classification layer. The EfficientNetB0 architecture has been added to the Tensorflow [22] and Keras [23] libraries.

This allows using CNN weights that are pre-trained on the large ImageNet dataset (which also contains car images).

In the EfficientNetB0 architecture, a layer corresponds to a computing unit that performs a specific image processing operation. Let's consider several layers of the CNN (Fig. 5) and explain their functions:

1. input_1 (InputLayer) – an input layer that accepts input images of 100x100 pixels with three color channels (RGB).
2. rescaling (Rescaling) – the layer scales the color channel values for the pixels of the input image to the range [0, 1]. This helps to prepare the data for further processing in the network and ensures the stability of the learning process.
3. normalization (Normalization) – this layer normalizes the input pixel color values to balance their distribution. It can perform different types of normalization, such as centering and scaling. Normalization helps to avoid problems with gradients during training.
4. stem_conv (Conv2D) – the layer is responsible for convolution of the input images. The convolution is applied with a kernel of 3 × 3 elements to highlight important features in the image.
5. stem_bn (BatchNormalization) – the layer performs batch normalization of the output values after convolution to stabilize the learning process and improve the convergence rate.
6. stem_activation (Activation) – the layer applies the activation function ReLU (Rectified Linear Unit) to the output values after batch normalization to introduce nonlinearity into the network.
7. block1a_dwconv (DepthwiseConv2D) – the layer performs depthwise convolution, i.e. independent convolution for each channel of the input image.
8. block1a_bn (BatchNormalization) – after depthwise convolution, this layer is used for batch normalization.
9. block1a_activation (Activation) – the activation layer applies a nonlinear function to the output values after batch normalization to preserve the nonlinear properties of the data.

Pre-training of CNNs greatly simplifies the learning process [24]. After the base model is loaded, the layers are frozen (fixed), which avoids overtraining on a small data set. Layer freezing is set with the parameter trainable=False. Next, a global mean pooling layer is added to the CNN model, which allows to collapse the spatial dimensions of the previous layer to a fixed-length vector. This helps to reduce the number of parameters and computational size of the model, which is especially important when training on limited amounts of data. After the global mean pooling step, two fully connected layers are added, which are used to classify the images into two classes: "car" and "non-car". The last layer uses softmax activation to select the class with the highest probability. Once the model is built, the compile method is called to set the training parameters, such as the optimizer, loss function, and metrics for evaluating the model.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 100, 100, 3       0         []
                             )]

rescaling (Rescaling)        (None, 100, 100, 3)       0         ['input_1[0][0]']

normalization (Normalization)  (None, 100, 100, 3)     7         ['rescaling[0][0]']

rescaling_1 (Rescaling)      (None, 100, 100, 3)       0         ['normalization[0][0]']

stem_conv_pad (ZeroPadding2D)  (None, 101, 101, 3)     0         ['rescaling_1[0][0]']

stem_conv (Conv2D)           (None, 50, 50, 32)        864       ['stem_conv_pad[0][0]']

stem_bn (BatchNormalization)  (None, 50, 50, 32)       128       ['stem_conv[0][0]']

stem_activation (Activation)  (None, 50, 50, 32)       0         ['stem_bn[0][0]']

block1a_dwconv (DepthwiseConv2  (None, 50, 50, 32)     288       ['stem_activation[0][0]']
D)

block1a_bn (BatchNormalization  (None, 50, 50, 32)     128       ['block1a_dwconv[0][0]']
)

block1a_activation (Activation  (None, 50, 50, 32)     0         ['block1a_bn[0][0]']
)
```

Figure 5: A fragment of the CNN model with the EfficientNetB0 architecture obtained from the tensorflow library.

The Adam optimizer was chosen, the binary_crossentropy loss function (since the task is binary classification), and the accuracy metric to evaluate the model's accuracy during training. The structure of the CNN model (Model summary) is shown in Figure 6.

```
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 4, 4, 1280)       4049571

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 128)               163968

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 4,213,797
Trainable params: 4,171,774
Non-trainable params: 42,023
```

Figure 6: A fragment of the structure of the CNM model with the EfficientNetB0 architecture.

The EfficientNetB0 architecture has a large number of parameters that are used to detect various features in images. In the case of the developed CNN model, the total number of

parameters is 4,213,797, so the model has a great potential power to detect complex dependencies in the input data.
A graphical interface for interactive adjustment of neural network parameters has been developed (Fig. 7). The interface was created using the tkinter library of the Python programming language. The developed interface makes it possible to select the location of the dataset, the number of epochs, the path of saving the model, etc.

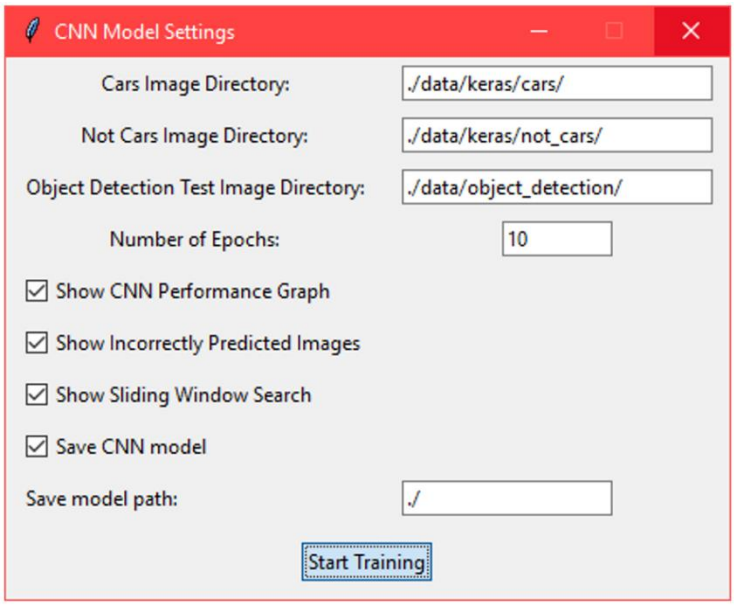

Figure 7: Graphical interface for setting up CNN parameters.

Also, using checkboxes, user can display a graph of model accuracy, show incorrectly recognized images during image validation, and implement car recognition using a sliding window as an example. This interface allows for quick and interactive change of various parameters of the neural network and the image recognition program itself. Just like the GUI, user can change all the parameters and run the neural network using the Command Line Interface (CLI).

5. Improvement of the Convolutional Neural Network Model

After the implementation of the basic CNN model with the EfficientNetB0 architecture, improvements were made. It is important to note that the efficiency of the basic model is already quite high due to preliminary training on a large ImageNet dataset.

5.1. Fine-tuning

Initially, the fine-tuning method was used to tune the base model with the EfficientNetB0 architecture, in which the layer weights were frozen (fixed). In the new model, some of the layers are unfrozen and subjected to fine-tuning along with the newly added layers on top. This allows the CNN model to learn on more specific image features using the knowledge from the previously trained weights. The last 20 layers of the base model were unfrozen by

setting the trainable parameter to True for each of these layers. The model was compiled using the Adam optimizer at a low learning rate (0.0001) for tuning.

The model was then trained on the dataset using the specified number of epochs, with a control sample used to prevent overfitting. The number of unfrozen layers, learning rate, and other hyperparameters were adjusted to meet the specific requirements of the dataset and the image recognition task. During fine-tuning, the model may be overtrained, especially if the amount of training data is limited. Therefore, it is important to monitor changes in the metrics and, if necessary, take timely measures to prevent overfitting, such as regularization or early stopping of training.

5.2. Early Stopping

Early stopping of training is used in the process of training the ANN model. This approach avoids overtraining by monitoring the losses on the validation dataset and stopping training when these losses stop decreasing or start increasing. For this purpose, the EarlyStopping callback from the Keras library was used. This callback monitors the losses on the validation dataset and stops training if the losses do not decrease for a certain number of epochs (the patience parameter). In addition, the restore_best_weights parameter allows to restore the model weights to those that gave the best result on the validation set. In this work, the callback object for early stopping was used with the parameters monitor='val_loss' (monitoring losses on the validation set), patience=3 (waiting for 3 epochs without improvement), and restore_best_weights=True (restoring the best model weights). This ensures that the training process will be stopped if the losses on the validation set do not improve within a specified number of epochs.

5.3. Learning Rate Scheduling

A study was conducted to adjust the learning rate using Learning Rate Scheduling. This approach allows to adjust the learning rate during training, which potentially leads to faster convergence and better model performance. To do this, the lr_scheduler function was defined, which reduces the learning rate by 10% every 10 epochs. Next, the lr_scheduler_callback object was created using the LearningRateScheduler callback, to which the learning rate scheduler function was passed. During model training, both callbacks responsible for early stopping and learning rate scheduling were passed to the callbacks parameter of the fit method. This ensures that during training, the learning rate will adapt according to the set schedule. This approach is a tool for optimizing the training process of neural networks. Reducing the learning rate over time can help to avoid delays in convergence and model overtraining. This allows the CNN to "rest" from sudden changes and ensures a more stable learning process. The use of Learning Rate Scheduling is especially useful in cases where complex data or large neural network architectures are used, where the learning process can be very sensitive to changes in the learning rate.

5.4. Improved Structure of the Convolutional Neural Network

In this improved structure of the CNN model, several important changes have been made to improve its performance and learning capabilities. Residual connections, a key element in

deep neural networks, are utilized. Residual connections enable the prediction of residual information that remains after passing through each layer of the network. This helps mitigate the vanishing gradients problem and contributes to faster training and improved model convergence. Adding residual connections between the layers of the model allows gradients to more easily propagate backward through the network during error backpropagation, leading to more effective learning.

The LeakyReLU activation function was also used, which is a modification of the standard ReLU activation function. In the standard ReLU function, neurons are not trained if their input value is less than zero. This can lead to stagnation problems and slower model training. Using LeakyReLU allows to pass negative values through the network with a small slope, which improves the gradient flow and helps avoid the problem of stalling. The alpha parameter specifies the slope value that adjusts the output of the LeakyReLU function for negative values. The choice of this parameter affects the speed and stability of training. A shortcut is created that is added to the output value after several fully connected layers. This allows the model to freely "bypass" some layers, which facilitates the learning process and allows the model to learn faster.

These changes in the model architecture are aimed at improving its learning capability and ability to avoid overfitting. The combination of the lagged links and the LeakyReLU activation function helps the model to detect complex dependencies in the data more effectively. In addition, the updated CNN structure adds another fully connected layer with the LeakyReLU activation function and Dropout layers, which helps to regularize the model and prevent overfitting. These additional layers help to moderate the complexity of the model, making it more generalizable and resistant to overfitting. They provide an additional layer of control over the propagation of gradients and the internal representation of data in the network.

6. Results

6.1. Training Results of the Basic Convolutional Neural Network Model

After the basic CNN model was built and compiled, it was trained on the training data for 10 epochs. After each epoch, the model's accuracy was evaluated on a control dataset (to avoid overfitting). The training took approximately 25 minutes. According to the results of the experiment on the control dataset, the model accuracy is 82.69%. This indicates that the CNN is able to cope with the task of detecting cars in images, but the accuracy is unsatisfactory. To improve the accuracy, an improved CNN model was used.

6.2. Training Results of the Improved Convolutional Neural Network Model

The improved CNN model with the EfficientNet architecture showed a significant increase in accuracy compared to the baseline model. The total number of parameters of the improved model is 4,451,685. When using images without rotation, the model training process lasted only 6 epochs (instead of the planned 10). This is due to the fact that the model stopped improving the results on the validation dataset, so additional training epochs were not needed. An important metric is the accuracy of the model, which reached

a value of about 99.77% on the validation dataset (Fig. 8). When using images with rotations, an extremely small training error of 2×10^{-4} was obtained for the validation dataset, and the model accuracy reached 99.98% (Fig. 9).

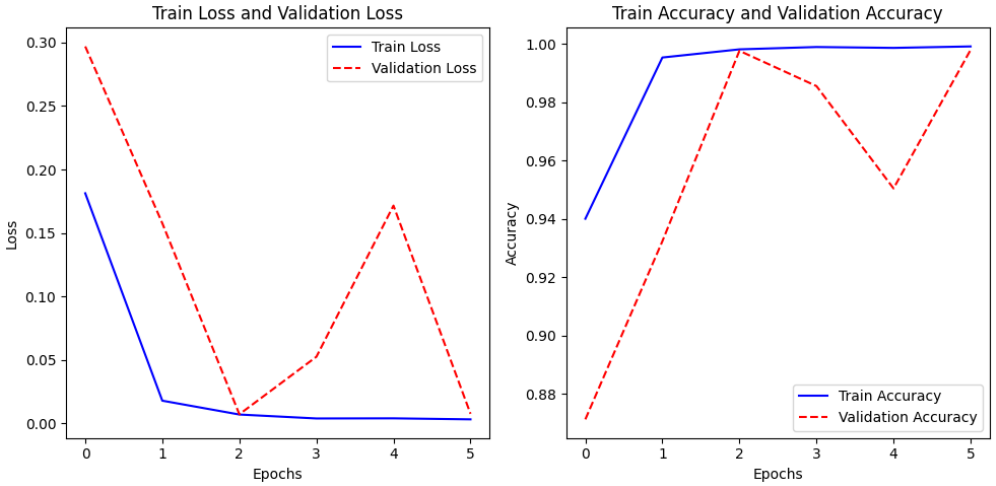

Figure 8: Graphs of training loss and accuracy for the improved CNN (image without rotation is used)

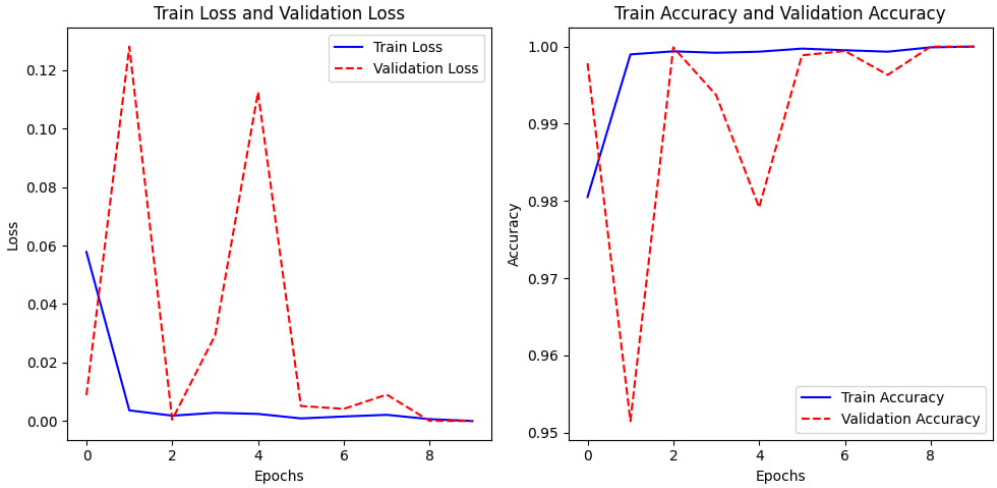

Figure 9: Graphs of training losses and accuracy for the improved CNN (image with rotations is used)

This indicates the high efficiency of the model, which allows for accurate classification of images with and without cars. The obtained accuracy exceeds the accuracy for the best analog program (99.63%), which was trained on this dataset [25]. This confirms the prospects of the developed CNN model.

7. Conclusion

A CNN model with the EfficientNet architecture has been developed for car image recognition. The CNN model is implemented using the Python programming language and the Keras library. The widely used Vehicle Detection Image Set dataset was used to train the model, which made it possible to compare the effectiveness of the proposed model with existing solutions. The structure of the developed model was improved: its parameters were adjusted, some layers were unfrozen, and stop conditions were added to avoid overtraining. These additional measures helped to improve the model's accuracy. Due to data augmentation, which was performed by randomly rotating the images, a high model training accuracy (99.98%) was obtained, which exceeds the training accuracy for the best analog (99.63%) on the used dataset.

The developed CNN model can be practically used for car image recognition.

References

[1] A. Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, Inc., 2019.

[2] N. K. Manaswi, Deep Learning with Applications Using Python, Apress, India, (2018). doi: 10.1007/978-1-4842-3516-4.

[3] K. He, X. Zhang, S. Ren and J. Sun, Deep Residual Learning for Image Recognition, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778. doi: 10.1109/CVPR.2016.90.

[4] G. Huang, S. Liu, L. Maaten and K. Q. Weinberger, CondenseNet: An Efficient DenseNet Using Learned Group Convolutions, in: Proceedings of the 2018 IEEE/CVF Conferenceon Computer Vision and Pattern Recognition, Salt Lake City, USA, 2018, pp. 2752–2761.

[5] T. Li, W. Jiao, L. Wang and G. Zhong, Automatic DenseNet sparsification, IEEE Access 8 (2020) 62561–62571.

[6] F. Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 1800-1807. doi: 10.1109/CVPR.2017.195.

[7] Tan Mingxing, V. Le Quocm, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, in: Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97 (2019) 1-11. URL: https://arxiv.org/pdf/1905.11946.pdf.

[8] R. Gonzalez, R. Woods, Digital image processing, Pearson/ Prentice Hall, New York, 2018.

[9] S. Russell, P. Norvig, Artificial Intelligence. A Modern Approach, Pearson Education, 2021.

[10] K. Kargin, Computer Vision Fundamentals and OpenCV Overview. URL: https://medium.com/mlearning-ai/computer-vision-fundamentals-and-opencv-overview-9a30fe94f0ce.

[11] Abdallah Wagih Ibrahim, Vehicle Detection. URL: https://www.kaggle.com/code/abdallahwagih/vehicle-detection-cnn-acc-99-3.
[12] M. Bayati, M. Çakmak, Real-Time Vehicle Detection for Surveillance of River Dredging Areas Using Convolutional Neural Networks, International Journal of Image, Graphics and Signal Processing (IJIGSP) 15 (5) (2023) 17-28. doi:10.5815/ijigsp.2023.05.02.
[13] O. Pavlova, A. Bilinska, A. Holovatiuk, Y. Binkovskyi, D. Melnychuk, Automated system for determining speed of cars ahead, Computer Systems and Information Technologies 3 (2023) 32-39. doi:10.31891/csit-2023-3-4.
[14] S. Balovsyak, Kh. Odaiska, O. Yakovenko, I.Iakovlieva, Adjusting the Brightness and Contrast parameters of digital video cameras using artificial neural networks, in: Proc. SPIE, Sixteenth International Conference on Correlation Optics 12938 (2024) 129380I-1 – 129380I-4. doi: 10.1117/12.3009429.
[15] S. Balovsyak, I. Fodchuk, Kh. Odaiska, Yu. Roman, E. Zaitseva, Analysis of X-Ray Moiré Images Using Artificial Neural Networks, in: IntelITSIS 2022: 3nd International Workshop on Intelligent Information Technologies and Systems of Information Security, March 23–25, 2022, Khmelnytskyi, Ukraine, CEUR Workshop Proceedings, 2022, pp. 187-197.
[16] S.V. Balovsyak, Kh. S. Odaiska, Automatic Determination of the Gaussian Noise Level on Digital Images by High-Pass Filtering for Regions of Interest, Cybernetics and Systems Analysis 54 (3) (2018) 662-670. doi: 10.1007/s10559-018-0067-3.
[17] V. Alto, Understanding the Inception Module in GoogLeNet. 2020. URL: https://valentinaalto.medium.com/understanding-the-inception-module-in-googlenet-2e1b7c406106.
[18] T. Hovorushchenko, V. Kysil, Selection of the artificial intelligence component for consultative and diagnostic information technology for glaucoma diagnosis, Computer Systems and Information Technologies 4 (2023) 87–90. doi: 10.31891/csit-2023-4-12.
[19] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou, A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects, IEEE Transactions on Neural Networks and Learning Systems 33 (12) (2022) 6999-7019. doi:10.1109/TNNLS.2021.3084827.
[20] A. Soni, A. Rai, An Efficient CNN Model for Automatic Diagnosis of Cardiomegaly from Chest Radiographic Images, International Journal of Image, Graphics and Signal Processing(IJIGSP) 15 (3) (2023) 81-96. doi:10.5815/ijigsp.2023.03.07.
[21] Vehicle Detection Image Set. URL: https://www.kaggle.com/datasets/brsdincer/vehicle-detection-image-set.
[22] TensorFlow. An end-to-end open source machine learning platform. URL: https://www.tensorflow.org.
[23] Keras. URL: https://keras.io.
[24] Chandra Mohan Bhuma, Ramanjaneyulu Kongara, A Novel Technique for Image Retrieval based on Concatenated Features Extracted from Big Dataset Pre-Trained CNNs, International Journal of Image, Graphics and Signal Processing(IJIGSP) 15 (2) (2023) 1-12. doi:10.5815/ijigsp.2023.02.01.
[25] Abhijit Singh, Vehicles identification. URL: https://www.kaggle.com/code/abhijitsingh001/vehicles-identification-val-acc-99-62/notebook.