STUDIA INFORMATICA
Nr 1-2 (23) Systems and information technology 2019

Kornel DOMERADZKI[1],
Artur NIEWIADOMSKI[1]

1 Siedlce University of Natural Sciences and Humanities
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

Object classification with artificial neural networks: A comparative analysis

DOI: 10.34739/si.2019.23.03

Abstract. Object classification is a problem which has attracted a lot of research attention in recent years. Traditional approach to this problem is built on a shallow trainable architecture that was meant to detect handcrafted features. That approach works poorly and introduces many complications in situations where one is to work with more than a couple types of objects in an image with a large resolution. That is why in the past few years convolutional and residual neural networks have experienced a tremendous rise in popularity. In this paper, we provide a review on topics related to artificial neural networks and a brief overview of our research. Our review begins with a short introduction to the topic of computer vision. Afterwards we cover briefly the concepts of neural networks, convolutional and residual neural networks and their commonly used models. Then we provide a comparative performance analysis of the previously mentioned models in a binary and multi-label classification problem. Finally, multiple conclusions are drawn, which are to serve as guidelines for future computer vision systems implementations.

Keywords. Object classification, neural networks, convolutional neural networks, residual neural networks

1. A Brief Introduction to Computer Vision

In order to understand the problem of image based classification one should acquaintance themselves with the field of computer vision and its terminology. Computer vision is an interdisciplinary field of science which strives to understand and process the data acquired from a source of spatial distribution of radiation. Every object that we can observe emits some kind of radiation. If a subject of interest does not emit any radiation, nothing can be observed or processed. That is why appropriate illumination is necessary for the targets of a computer vision system. In order for the program to process the electromagnetic radiation we need a contraption capable of capturing said data. In the majority of situations such a task is handled by optical lenses however it could also be an imaging optical spectrometer, an x-ray tomograph or even a microwave dish. Those devices are commonly called cameras. In order for one of the aforementioned devices to supply the information to a processing unit, the captured radiation must be converted into digitized data. That process is handled by the sensors. They transform the received radiative flux density into a desired signal which then can be used for further processing [1]. Said processing often means presenting higher-dimensional data, extracting desired features or categorizing the objects detected in the image into classes which is the main subject of this article.

Originally in computer vision systems, image classification and object detection have been approached by extracting hand crafted features. Most systems used SIFT (Scale-Invariant Feature Transform [2], or SURF [3] (Speeded Up Robust Features methods which find features in the Gaussian scale space (particular instance of linear diffusion). However, the problem with that solution was that blurring an image resulted in not respecting the natural boundaries of objects in the same degree an unprocessed image would, which is why some other systems adopted Kaze [4] instead. Feature extraction remains however an imperfect solution to the problem since it tends to struggle with delivering the correct results for certain cases, mostly due to the possible diversity of appearances of the target, illumination conditions or different types of backgrounds. This has eventually led to a huge interest in appliance of different solutions in this problem which are neural networks - mainly speaking convolutional and residual neural networks.

2. A Brief Overview of Artificial Neural Networks in Computer Vision systems

Artificial neural network is a concept of imitating the structure of a brain which has emerged for the first time in 1943 after the introduction of simplified neurons by McCulloch and Pitts [5]. Neurons in that article were presented as models of biological neurons and as conceptual

entities for circuits which were to perform certain computational tasks. That concept had however many deficiencies which were described in Minsky and Papert's book *Perceptrons* [6]. That made most researchers leave the field. Interest in that concept has re-emerged after 40 years - in the early eighties due to the discovery of error back-propagation and the continuous hardware advancements at the time.

The modern artificial neural network is usually defined as a computational model with particular properties such as the ability to adapt or learn, to generalize, or to cluster or organize data, and which operation can be based on parallel processing [7]. The aforementioned models can be visualized as a continuous net made up of layers, neurons and connections between them. Layers can be divided into three basic categories: input, hidden and output layers, as presented in Fig. 1.

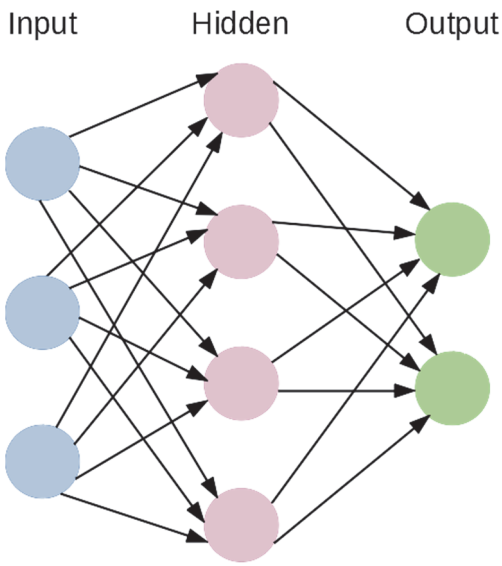

Figure 1. Structure of a basic neural network. Source: own study.

Each neuron in all of the aforementioned layers accepts input from one or more sources and produces a single value which can be output to multiple destinations. The source and destination of the neuron's data however varies depending on which layer it resides in. Input layer's neurons accept values supplied by the program and output their values to all the neurons of the first hidden layer. Hidden layers' neurons accept input from each single neuron from the previous layer, multiply them by their corresponding weight, add the results of multiplication up, add a bias to that if defined and then they use the calculated value in a chosen activation function (see Eq. 1). The output of the activation function is the value that the neuron will hold. Those operations are the connections that connect neurons from the previous layer to the next one. That value after being calculated is then output to the next layer. That process continues on, until the value reaches the output layer.

$$a_n^m = f\left(\sum_{i=0}^{l} w_i^{(m-1)} a_i^{(m-1)} + b\right) \quad (1)$$

where:

a – a value hold by a given neuron
m – the number of the layer (depth)
w – the weight value
l – the amount of neurons in m-1 layer
b – the value of bias of a given neuron connection

In the output layer values held by the neurons are calculated the same way they were in the hidden layers. Output however is not altered in any way. Internal state of the neuron serves as both the output and the final result of the neural network's operation.

Due to their structure, classical neural networks are a very suitable solution to a lot of problems in the computer vision field. They have however a major flaw which slows down the calculations in both the learning and operation significantly. That flaw is each neuron being fully connected to all the neurons from the previous layer. That approach creates an unnecessarily huge amount of parameters that have to be trained which often leads to prolonging the training time and overfitting the model. That is one of the reasons why convolutional neural networks have recently become so popular.

Convolutional neural networks are designed with the idea of working on image data. They are mainly used for classifying the image into specific classes. They have neurons arranged in 3 dimensions: width, height and depth (it is important to note that in this type of network depth refers to the amount of channels in an image rather than the depth of layer). That means that an input image made of a matrix of 32 pixels by 32 pixels with 3 colour channels (red, green and blue) would have these dimensions: 32x32x3 (width, height, depth respectively).

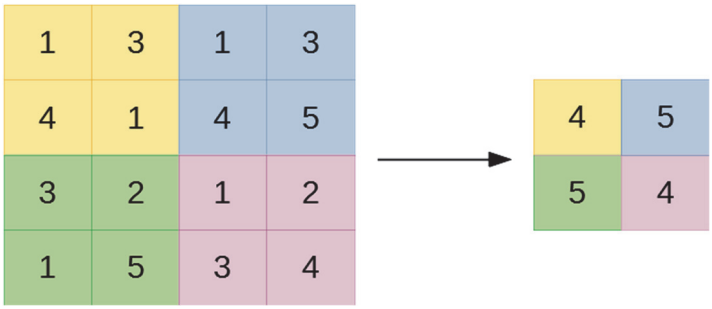

Figure 2. Visualization of the pooling process. Source: own study.

A typical network of this type usually consists of four types of layers: Convolutional, Pooling, Rectified Linear Units (ReLUs) and Fully connected layers. Convolutional layers

consist of a set of learnable filters. All of the filters are small spatially however they extend through the full depth of the input volume. The data that the layer works on constantly changes since the filters are to be applied to multiple cutouts of the image - in other words we can say that we are sliding (convolving) our filter frame over the image and compute the dot products with the given inputs. As we convolve over the width and height of the input volume we produce a two dimensional activation map that signifies the possibility of a given feature's presence. As the network learns it determines useful patterns for itself. Those patterns usually are visual features such as edges that are slanted at a specific angle, blotches of some colour or even complex patterns such as honeycomb or wheel patterns. Pooling layer is a type of layer that is often put in between convolutional layers. Their main objective is to reduce the amount of parameters and computation in the network and as a result of that also to control overfitting. This layer acts independently on every depth slice of the input and resizes it spatially, using the MAX operation (see Fig. 2). The result of such an operation is that the input volume which could be the image data or the feature map will be scaled down. Most common practice is using pooling layers with filters of size of 2x2 with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.

Rectified Linear Units layers are also responsible for transforming the feature map. They take in the values from either the convolution layers or pooling layers and transform negative values to zero. The last type of layer is the fully connected layer which in terms of the structure is the same as the hidden or output layers from the classical neural networks. Usually they are used for the output of the network. Each neuron in such an appliance holds the possibility of the object on the image being classified into the specific class.

CNNs however just like any other solution have a certain flaw which made using them in some cases quite problematic. That problem occurs in situations where we want to enrich the levels of the features by adding more layers to a given architecture. That however often counter intuitively results in a model which has a higher training error than the model with less layers.

That problem can be addressed by creating residual neural networks with the use of residual blocks. Residual block is a concept of putting a skip connection before a linear layer and before a ReLu layer. That skip connection transfers the values from a certain layer to a further point in the network and then an addition operation between them and the values from a linear layer is performed (see Fig. 3). This approach according to the *"Deep Residual Learning for Image Recognition"* [8] paper may allow creation of networks that are up to 8 times deeper and much more accurate.

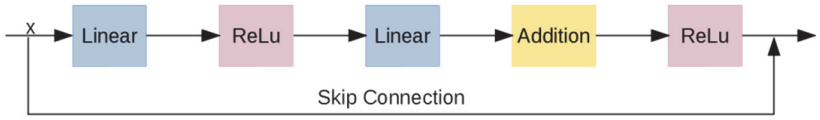

Figure 3. Residual Block's Structure. Source: own study.

3. Comparative Performance Analysis of Artificial Neural Network Models

The goal of this comparative performance analysis is to measure how well all of the three chosen architectures perform against one another in two chosen scenarios representing very common use cases. The comparison criteria are: compilation time, training time, and classification accuracy.

3.1. Overview of the experimentation methodology

All of the architectures which are taken into consideration in this experimentation consist of three main components. They are respectively: the resizing layer, base feature extractor and the classifier. The resizing layer is a simple keras provided layer which resizes the received input into a vector containing data on a 75 by 75 pixel image with the use of bilinear interpolation. For the feature extractor component three feature extractors are taken into consideration: MobileNetV2 [9], InceptionV3 [10] and InceptionResNetV2 [11]. MobileNetV2 was chosen due to its popularity in mobile and low-spec device applications. InceptionV3 and InceptionResNetV2 were chosen as potential alternatives for higher-spec device appliances in which higher evaluation accuracy is desired. Each of the aforementioned extractors are very deep which can significantly increase the training time. That is why we have decided to load the pre-trained weight values provided by the keras library prior to the start of the training which should result in shorter training times. The last component is the classifier. It consists of a single global average pooling 2D layer and a simple densely connected layer. The output values of the neurons present at the end of the classifier point at the class into which the object is classified.

The training process of each scenario results in three models. The script trains all of the architectures for 20 epochs. After each epoch the validation accuracy is checked and the data is stored in training history for later processing. The precaution taken to prevent overfitting is cross-validation which is used with the data split into validation and training data in such ratio: 80% of each batch is used for training and the remainder is used for validation. The used loss function is sparse categorical cross entropy. The chosen optimization technique is the RMSprop algorithm and the base learning rate is 0.0001.

Two chosen scenarios in which the networks are tested are meant to simulate very common scenarios in which those architectures could be used. The First scenario is a simple binary classification problem of classifying a cropped image of a parking place into two classes - the first one being *an occupied parking place* and the second one being *a free parking place*. For training, testing and benchmarking all of the models are trained on cropped images of the CNR Parking Dataset [12, 13]. The second scenario is a multi-label classification problem. In this case the CIFAR-10 [14] dataset is used. That dataset contains images of 10 mutually exclusive classes representing vehicles and animals which vary in shapes and sizes which should verify the ability of the networks to detect distinct features of objects in more diverse classification problems.

The time measurements were taken using the standard python library and the frameworks used for the implementation of the benchmarks are tensorflow 2.3.0 and keras 2.4.3. Version of the used Python interpreter is 3.8.5.

The experiments have been performed using a PC with these specifications: MSI B450 Tomahawk motherboard, AMD Ryzen 7 2700X CPU, Asus RTX2070 8GB Turbo GPU, 4 sticks of 4GB DDR4 2666MHz CL15 RAM running Windows 10 Pro version 2004. The benchmarks utilize the support of the CUDA toolset V10.1.243.

3.2. Binary Classification Problem

The binary classification problem is often considered an easy classification problem because of the sole amount of classes to which the objects can be categorized. That however does not always have to be true. The chosen parking occupancy test is an interesting case of this problem due to the variety of objects' sizes, shapes and colours. The other complication of this classification is how the weather affects the objects. Depending on the weather it might be hard to detect edges of some vehicles due to the sunlight's reflection or the raindrops might distort the received image from the camera. The CNR Parking dataset contains images with all of the aforementioned circumstances.

Each of the models are trained on a dataset of over 3000 cropped parking images. Every image is of 75 by 75 pixels size. As it was mentioned before the images cover most of the problems that the network might have to deal with. The only exception is snowy conditions which are not present in the CNR parking dataset.

3.3. Multi-label Classification Problem

The multi-label classification problem is very common in modern artificial intelligence powered systems. It is present in appliances such as classifying the types of cells from electronic microscope scans and in classifying races of animals in an image. Currently this problem is definitely one of the most important problems in computer vision. One of the biggest obstacles for software developers when implementing a neural network for handling this problem is however picking the correct model that suits the performance and accuracy needs.

Every model in this experimentation is trained to recognize one of 10 fundamentally different objects. All images in the dataset cover large angle variations and background clutter. CIFAR-10 dataset consists of 60,000 32x32 pixel images. Each class has 6000 images representing it. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

3.4. Experimental Results

This section discusses the data received in the aforementioned experimentation. We have decided to measure the time of compilation, training, and evaluation and the accuracy of the models over the epochs of training and highlight the evaluation capabilities of the resulted model. Each of these data is imperative to create a well performing classification system. Compilation time is the time it takes for the framework to compile a model and prepare it for evaluation. It is often overlooked however it tends to be important when running a network of small internet of things devices which have to use a different model depending on the circumstances. The second measured information is the time of training. It is crucial for a software engineer to find a model which can be trained to a certain accuracy level in as little time as possible. Evaluation time and accuracy of the final models are the most important pieces of data in this experimentation. They will tell us how quickly a model will finish its execution and how precise its classification will be. Most likely depending on that we might be determining which one is the most suitable for our application.

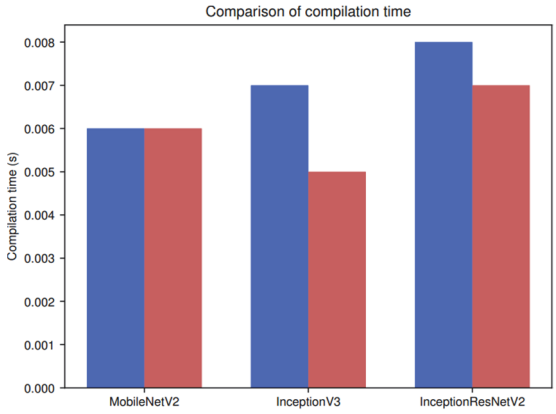

Figure 4. Average compilation time of Models. The blue bars represent the binary classification models' data and the red ones represent the multi-label classification models' data. Source: own study.

Figure 4 apparently shows that the binary classification model which compiled the fastest was InceptionV3. It compiled in around 0.0060 seconds. MobileNetV2 compiled in 0.007 seconds and InceptionResNetV2 compiled the slowest - in 0.0080 seconds. In the second scenario the fastest to compile was InceptionV3 which took only around 0.0050 seconds. MobileNetV2 compiled in 0.006 seconds and InceptionResNetV2 compiled the slowest - in 0.007 seconds. Models which have more layers take longer to compile.

In Figure 5 one can see that the fastest binary classification model to finish training is MobileNetV2 which took only 71.18 seconds to train. InceptionV3 had to be trained for 80.66 seconds and the slowest one to train was InceptionResNetV2 which had to be trained for 144.68 seconds. Based on this data we can assume that MobileNetV2 is the quickest model to train of them all.

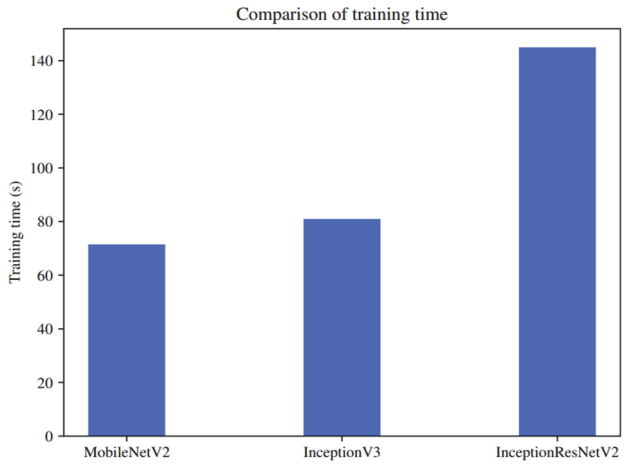

Figure 5. Training time for binary classification models. Source: own study.

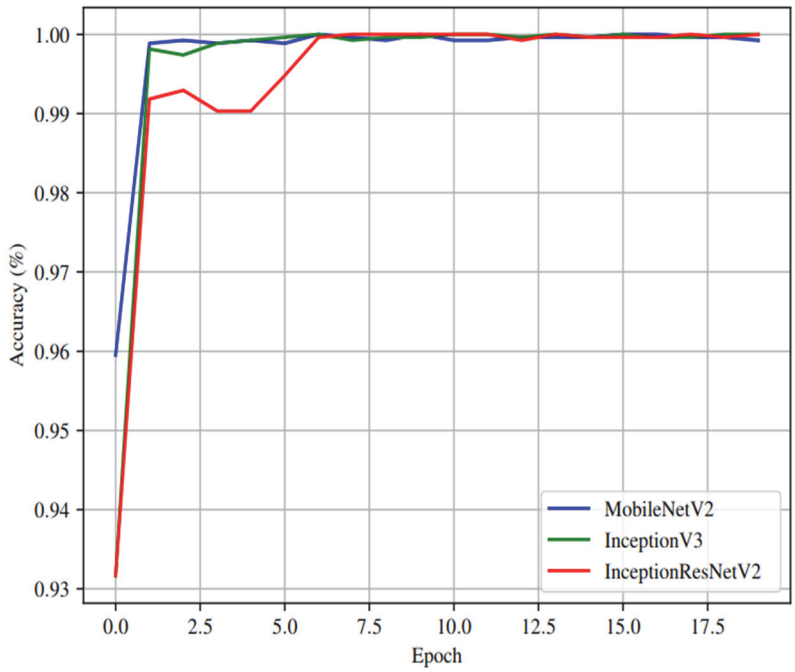

Figure 6. Evaluation accuracy of binary classification models after each epoch. Blue: MobileNetV2, green: InceptionV3, red: InceptionResNetV2. Source: own study.

After analysing data depicted in Figure 6 we can tell that the fastest binary classification model to reach its optimal point is MobileNetV2 which reached it after just one training. InceptionV3 reaches it after 5 epochs and InceptionResNetV3 reaches it after 6 epochs. Based on this data, training after 6 epochs should result in little to no difference for all models.

Figure 7. Training time for multi-label classification models. Source: own study.

Figure 7 shows that the fastest multi-label classification model to finish training is MobileNetV2 which took 1632.19 seconds to train. InceptionV3 had to be trained for 2605.42 seconds and the slowest one to train was InceptionResNetV2 which had to be trained for 4390.17 seconds.

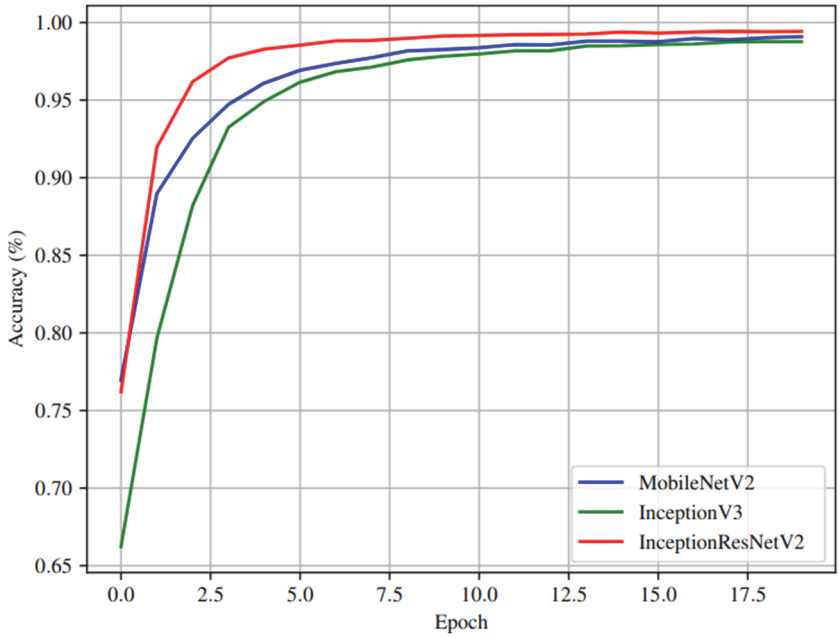

Figure 8. Evaluation accuracy of multi-label classification models after each epoch. Blue: MobileNetV2, green: InceptionV3, red: InceptionResNetV2. Source: own study.

Figure 8 shows that the accuracy over training in all of the multi-label classification models increases at a much slower rate than in the binary classification problem. InceptionResNetV2 seems to achieve the optimal point around the 13th epoch and doesn't improve much after it while the MobileNetV2 and InceptionV3 keep improving up to the 20th epoch. Faster training time of InceptionResNetV2 is most likely the result of that network being optimized for complex multi-label classification problems and the amount of residual blocks present in its architecture.

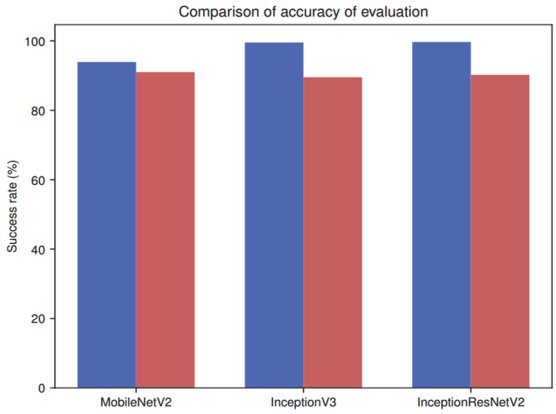

Figure 9. Evaluation accuracy of the received models. The blue bars represent the binary classification and the red ones represent the multi-label classification. Source: own study.

The results depicted in Figure 9 show that in binary classification models it is apparent that deeper nets are more accurate. The received accuracy for each of the models is respectively: 93.89%, 99.55% and 99.70%. There is a big 6% accuracy gap between a light-weight MobileNetV2 and deeper models such as InceptionV3 and InceptionResNetV2. In this scenario the accuracy is relative to how deep each of the models are. In multi-label classification on the other hand that accuracy gap lowers significantly. Accuracy for each of the models is respectively: 91.01%, 89.56% and 90.24%.

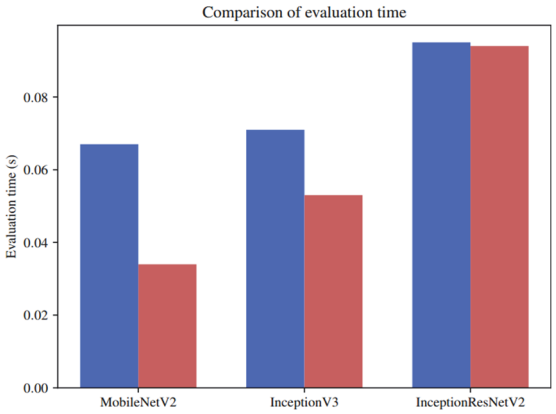

Figure 10. Evaluation time of a single image by the models. The blue bars represent the binary classification and the red ones stand for the multi-label classification. Source: own study.

In Figure 10 we can see that in both scenarios deeper models take much more time to evaluate an image. In the binary classification the fastest to evaluate an image is MobileNetV2 which can classify an object in around 0.067 seconds. InceptionV3's average is 0.071s and InceptionResNetV2's is 0.095s.

The fastest model to finish its execution in multi-label classification is MobileNetV2 which can do it in around 0.034 seconds. InceptionV3's average is 0.053s and InceptionResNetV2's is 0.094s. Surprisingly all of the models in the second scenario take much less time to evaluate an image. That could be due to the feature extractors being optimized for multi-label classification problems rather than binary classification.

4. Conclusions

The analysis of the experimentation results allowed us to draw some conclusions. The first one is that compilation time in both cases for each of the models can be considered insignificant since it is lower than a tenth of a second. The delay of the script's startup most likely is caused by the tensorflow API loading and processing the Nvidia's CUDA dynamic libraries.

Training time on the other hand, strongly depends on the type of the model and the number of classes into which the objects are to be classified. In the case of binary classification most models reach their peak accuracy relatively quick - in the worst case scenario after six epochs. Ten class classification problem however requires over 23 times more time than a binary classification problem to perform 20 epochs of training. Just as in binary classification they do reach a peak accuracy point before 20 epochs. That point is reached after the thirteenth epoch and training after it results in little to no differences in accuracy.

Discrepancy of evaluation accuracy results is apparent in the binary classification problem. MobileNetV2 is the least accurate model while the residual network models thrive due to them being on average 6% more accurate. On the other hand in the other scenario all of the models perform very well and the difference in successful classification between them is only around 1%. It is however apparent that residual networks are able to get higher accuracy results with less training. The results of evaluation time with the use of each of the models follows a similar pattern in both cases. MobileNetV2 is the fastest following InceptionV3 and InceptionResNetV2. The residual networks take more time to evaluate an image.

Based on the drawn conclusions we can safely assume that MobileNetV2 is indeed the best choice for a low-spec or mobile devices due to its quickest evaluation time with relatively little trade off for accuracy. If there is a need for training the model on a mobile device this architecture will be the fastest to train as well. Those traits make it a great choice for use in object detection frameworks in mobile camera systems. InceptionV3 and InceptionResNetV2 are on the other hand more accurate and demand more computing power. They tend to be twice or thrice as slow as the aforementioned option. Their unique characteristic of being able to be trained quicker than MobileNetV2 is due to them both being deeper networks with more residual blocks. That can be very useful for some specific scenarios where the net is supposed to learn new patterns. Those given architectures can be definitely used for appliances such as object detection in an automated car's camera.

References

1. B. Jahne (Ed.): Computer vision and applications: a guide for students and practitioners. Elsevier, 2000.

2. D. G. Lowe: Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2):91-110, 2004.

3. H. Bay, A. Ess, T. Tuytelaars, and L. van Gool: Speeded-up robust features (surf). Computer vision and image understanding, 110(3):346-359, 2008.

4. P. F. Alcantarilla, J. Nuevo, and A. Bartoli: Fast explicit diffusion for accelerated features in nonlinear scale spaces. In British Machine Vision Conf. (BMVC), 2013.

5. W. S. McCulloch, W. Pitts: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115-133, 1943.

6. M. Minsky, S. A. Papert. Perceptrons: An introduction to computational geometry. MIT press, 2017.

7. B. Kröse, P. van der Smagt: An introduction to neural networks. 1993.

8. K. He, X. Zhang, S. Ren, and J. Sun: Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770-778, 2016.

9. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen: MobileNetV2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4510-4520, 2018.

10. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna: Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818-2826, 2016.

11. C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi" Inception-v4, Inception-Resnet and the impact of residual connections on learning. arXiv preprint:1602.07261, 2016

12. G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo: Deep learning for decentralized parking lot occupancy detection. Expert Systems with Applications, 72:327-334, 2017.

13. G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo: Car parking occupancy detection using smart camera networks and deep learning. In Computers and Communication (ISCC), 2016 IEEE Symposium, pp. 1212-1217. IEEE, 2016.

14. A. Krizhevsky, G. Hinton, et al.: Learning multiple layers of features from tiny images. 2009.