symmetry

MDPI

Article

An Automated Training of Deep Learning Networks by 3D Virtual Models for Object Recognition

Kamil Židek *, Peter Lazorík, Ján Piteľ and Alexander Hošovský

Faculty of Manufacturing Technologies with a seat in Presov, Department of Industrial Engineering and Informatics, Technical University of Kosice, Bayerova 1, 08001 Prešov, Slovakia; peter.lazorik@tuke.sk (P.L.); jan.pitel@tuke.sk (J.P.); alexander.hosovsky@tuke.sk (A.H.)

* Correspondence: kamil.zidek@tuke.sk; Tel.: +421-903-137961

Received: 15 March 2019; Accepted: 2 April 2019; Published: 5 April 2019

Abstract: Small series production with a high level of variability is not suitable for full automation. So, a manual assembly process must be used, which can be improved by cooperative robots and assisted by augmented reality devices. The assisted assembly process needs reliable object recognition implementation. Currently used technologies with markers do not work reliably with objects without distinctive texture, for example, screws, nuts, and washers (single colored parts). The methodology presented in the paper introduces a new approach to object detection using deep learning networks trained remotely by 3D virtual models. Remote web application generates training input datasets from virtual 3D models. This new approach was evaluated by two different neural network models (Faster RCNN Inception v2 with SSD, MobileNet V2 with SSD). The main advantage of this approach is the very fast preparation of the 2D sample training dataset from virtual 3D models. The whole process can run in Cloud. The experiments were conducted with standard parts (nuts, screws, washers) and the recognition precision achieved was comparable with training by real samples. The learned models were tested by two different embedded devices with an Android operating system: Virtual Reality (VR) glasses, Cardboard (Samsung S7), and Augmented Reality (AR) smart glasses (Epson Moverio M350). The recognition processing delays of the learned models running in embedded devices based on an ARM processor and standard x86 processing unit were also tested for performance comparison.

Keywords: convolutional neural networks; virtual 3D model; pattern recognition

1. Introduction

The modern approach to analyze 2D images (object detection) is based on convolutional neural network (CNN, or ConvNet) models, which are a part of deep learning techniques. The standard artificial neural network (ANN) differs from convolutional networks mainly in the structure of hidden layers of neurons and a significantly higher number of neurons per layer [1]. The standard ANN is suitable for solving more general tasks, but preparation of data for input layers is time-consuming and more complicated.

The main advantage of the deep learning approach with CNN is the simplification of the extraction of image data, which has been done before by standard image processing algorithms (thresholding, contouring, segmentation) [2] with the combination of data mining algorithms (clustering, classification) [3]. Many different types of CNNs have been developed, which differ in the structure of a hidden layer, for example, R-CNN, MobileNet, Alexnet, Inception, GoogleNet, etc. The hidden layers are not only important in neural networks, but also in the dynamics over general complex networks [4]. It is possible to put a 2D bitmap image of the object onto the input layer of the

CNN directly and to execute training, evaluation, and testing, so complicated image processing to numeric values is unnecessary.

1.1. Problem Specification and Motivation

Convolutional networks need a large quantity of object images for the training process. They can be used for multiple object detection in a single image and detect partially visible objects, for example, inside of assembly or construction nodes [5]. If a training set contains object samples from every point of view, the recognition process can be very reliable during detection. Due to these properties, CNNs are suitable for industrial part recognition or part identification inside of construction nodes.

Dynamic object recognition is necessary in virtual or augmented reality devices used in industrial tasks with the Industry 4.0 concept. Some of these tasks can be solved using CNN models implemented into VR/AR devices for assisted assembly. This approach is a current trend for very variable production, where it is not possible to establish standard automation with production lines and industrial robots. The assisted assembly process needs smart glasses, which can help to teach and check new workers in a very short time without additional staff. The main feature of the automated assisted process is reliable real time part localization and recognition. If production is very variable, with many variants of products or small production batches, the implementation process for CNN takes a long time, mainly due to the manual preparation of samples for training.

1.2. Research Idea and Benefits

Our idea of how to make the implementation process for CNN faster is to teach the CNN model using 2D samples generated automatically from virtual 3D models. Convolutional neural networks with deep learning can work reliably for object recognition, but the problem is manual preparation of input data for the learning process. A very large quantity of input samples need to be prepared, usually several hundred for one object, which is a prerequisite for reliable learning of the object groups. If we need a large number of objects to recognize, this introductory process can be extended to several weeks of monotonous work. The object must be captured with different angular/translation variations and partial overlapping by other objects and also with different backgrounds and materials. The replacement of real image samples with 3D virtual models for CNN training significantly accelerates this monotonous work, especially when many variations of materials and backgrounds are needed.

We need to specify a suitable export format that is universal for all major 3D design software: Pro Engineer Creo, Data Assault CATIA, Autodesk Inventor, Solid Edge, Siemens NX, etc. The standard export formats for 3D design software are: dxf, stl, vrml, obj, iges, step, and 3ds. The most precise format used for rapid prototyping is stl, but this format does not retain information about the material and color of parts or the assembly product. We chose the ".obj" WaveFront format. This format has an economical grid generation, it preserves color information after export, and it is supported by the software that we use to create samples. This format was primarily developed for 3D animation software, but it is also supported by industrial 3D design software.

2. Relevant Previous Works

The early research on pattern recognition in our department aimed to achieve recognition of industrial parts in production lines and component surface errors using standard algorithms based on machine learning [6]. Classification technics (support vector machines, normal Bayesian classifier, K-nearest neighbor, gradient boosted trees, random trees) were implemented to embedded systems for surface error recognition [3,7]. The next research was provided with CNN models for the recognition of standardized industrial parts (hexagon screw/nut and circular hole assembly element) trained by real image input [8].

Generally, 3D models are usually available from the design process before the production of the product starts. This means that 3D models of assembly products (respectively parts) are available as input data for CNN training. The research on virtual 3D model CNN training without any automation

can be summarized as follows: Research on the training and testing of CNN detectors using virtual images [9], recursive analysis of 3D models via convolution networks [10], CNN training using 3D models [11], learning of depth detectors by 3D models [12], analysis of 3D objects via CNN [13], and CNN trained with 3D rendered images [14].

An important task is to make a bridge between the 3D design software and teaching software (CNN frame-work). The research related to 3D design can be found in various papers, e.g., about parameter enhancement in a 3D model [15], 3D CAD (Computer Aided Design) lightweight representation [16], or design process automation [17]. The automated sample generation from 3D models can be simplified using a web interface. The principle of how to automatically manage generated production documentation is described in [18].

Besides automated input data preparation for CNN training, automated image analysis is also important. Some principles of this process were described in [19]. An interesting case study on the recognition of mark images using deep convolutional neural networks was published in [20]. The use of CNNs to detect anti-3D models for safe 3D printing by training D2 vectors, which were constructed from the D2 shape distribution of 3D models, was described in [21].

The learned CNN model can also be used in VR/AR devices for object recognition. These devices, together with collaborative robots, can improve the manual assembly process (known as an assisted assembly) [22]. It is important that the structural and semantic data from 2D plans is extracted for the creation of a virtual environment [23]. The improved augmented reality registration methods presented in [24] and tracking algorithms using MEMS (Micro Electro Mechanical System) devices [25] can help to minimize the chance of a robot colliding [26] with a human in the assembly process.

3. Proposed Work

This article presents a new methodology for speeding up the CNN training process based on automated generation of input sample data for learning without any monotonous manual work. All tasks, such as a part angle/position definition, background and material change, and object detection position (SSD), can be automated by scripting language. In this way, it is possible to shorten the time of sample preparation (sufficient number of samples is between 100 to 300 per object) to a few minutes in the automated cycle instead of hours without automation. The input 3D object model is usually available before this process from 3D construction software. The methodology and the software/hardware implementation with recognition reliability verification is described in the next chapters.

3.1. The Principle of 2D Sample Generation

The principle scheme of automated input data generation from a virtual 3D model for a convolutional network is shown in Figure 1.

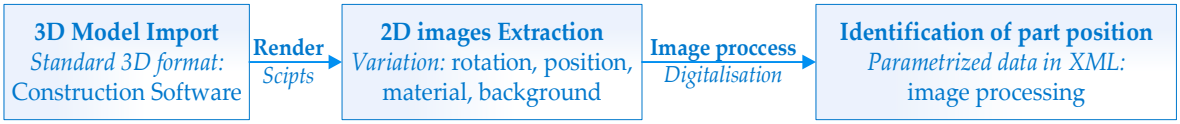


Figure 1. Scheme of an automated 2D sample generation from a 3D model.

The detailed procedure of the 2D data training samples' generation consists of several steps, which are fully automated by scripts:

- Import the 3D model from any 3D CAD software (CATIA, CREO, Autodesk Inventor, etc.) to 3D rendering software with integrated scripting language.
- Generate 2D sample images with different object rotations/position, textures, and backgrounds, including standard views.

- Generate the object bounding box in the 2D image with basic parameters (position/dimension) for part localization by standard image processing techniques (Python OpenCV library) used for single shot detection (SSD) during training.
- Generate the text description file (XML—eXtensible Markup Language) with the following basic image parameters: Name of file, size and position, and image resolution.
- Separate the random samples into two groups: Training set (80%) and validation set (20%).

The main principle of 2D object extraction from 3D models is part rotation and translation in the scene. The object movement is described by the rotation matrix with the center point of gravity given by Formula (1) and the translation matrix given by Formula (2). We rotated the objects around two axes (x and z) by a 20° increment. The validation samples were generated with a random angle of rotation and random position change.

$$R_y = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \; R_z = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where:

R_y, R_z—matrix of virtual 3D rotation of part.
β, γ—rotation angle for each additional generated view.

$$T_{xyz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{2}$$

where:

T_{xyz}—matrix of the virtual 3D translation of a part.
t_x, t_y, t_z—random translation in pixels for each additional generated view.

The introduced methodology was implemented into Python automated script, which is universally usable for any 3D design CAD software that exports to the obj file format. We used the obj export from Autodesk Inventor software for testing. Rendering of the 2D images was realized by Blender 3D software. This rendering and modeling software is open source and it provides an internal Python language. In this way, we could directly control the current object scene angle, position, material, and background by common API (Application Programming Interface) commands.

After 2D sample generation, the part size and its position inside of the image needed to be localized. This task was realized by the Python wrapper of the OpenCV library. OpenCV detects the width, height, and X, Y position of the part in a specialized temporary frame with a black background and white material. These data are written into the XML file format, which is generated for every frame file and it includes: The name of the image file, type of the object, the image height/width in pixels, the object position, and its size in pixels. The TensorFlow framework from Google (GPU variant—Graphical Processing Unit) was selected for transfer learning of pretrained CNN models. For the experiments, we selected two models: Precise, but more complicated model, Faster R-CNN Inception V2 (52 MB); and the simpler and faster model, MobileNet V2 (19 MB), pretrained on a coco dataset. The new generated input data were trained by the transfer learning technique. The trained models were next frozen and exported to OpenCV for recognition processing delay experiments in embedded devices. The software implementation procedure and transfer learning of a convolutional network is shown in Figure 2.

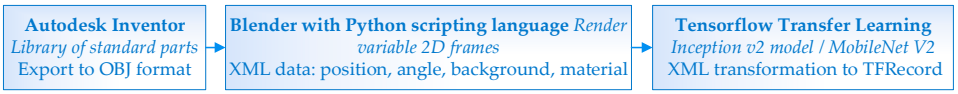


Figure 2. The detailed implementation scheme for creating 2D input data from 3D virtual models.

3.2. Input Images Used for the Training Process

The input samples were generated in the PNG format; 36 frames with different materials and an adequate background (108 samples for one object) were generated for each variant of the components. The following combinations based on the brightness of textures were chosen (Figure 3):

- Chrome part material—wood table base.
- Brass part material—base steel plate.
- Steel material—polished rock workplace.

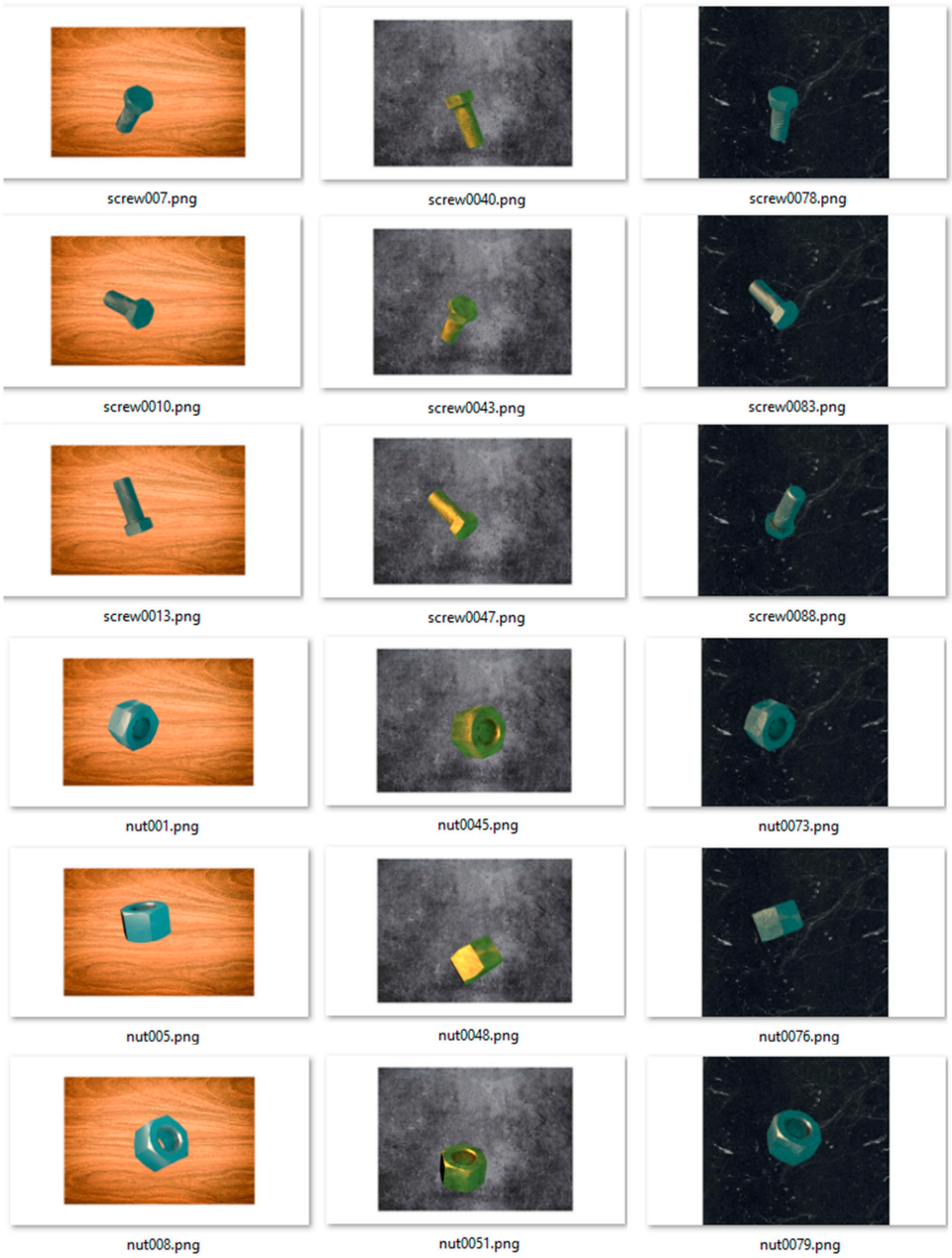


Figure 3. An example of the image set used as teaching samples (nut/screw).

A material texture of the object has to be combined with a suitable background texture to ensure good visibility for the teaching process. This selection process is currently handled manually, but it will be automatized in the future by histogram analyses for both textures. The background material combination with the object texture was selected with the condition of brightness difference maximization (high brightness—wood, middle brightness—steel, low brightness—rock). This selection of the background/object texture combination provides minimal sensitivity of the trained CNN model to changes in the environment's light condition (it recognizes dark objects on a white background and in reverse). The teaching samples simulate the standard parts (nuts, screw, washer) with proportional dimensions for the M12 hexagonal screw size. Two sets (groups) of samples were created for the teaching process, and the first bigger set (80%) was used for training and the second one was used for an evaluation of the teaching progress (20%). The sample preparation set did not contain samples that overlapped because this was our first experiment, which should confirm that virtual samples are suitable for the teaching of a convolutional model and that they can reach comparable results to real photo input samples from produced parts.

The generation process for the input samples used an optional rendering engine (Cycles), which is part of the Blender software. The default internal rendering engine from Blender was not suitable for rendering industrial parts (it creates a matte surface). All images were generated with the same resolution of 960 × 540, which is sufficient for training by the Inception V2 and MobileNet CNN model. Generated input images are shown in Figure 3.

After sample generation, the XML file was created for each part. The XML file contains data regarding the image size, part type, and object location within the scene.

The graphical process to virtualize standard industrial parts with an explanation of the XML generation from a 2D image is shown in Figure 4.

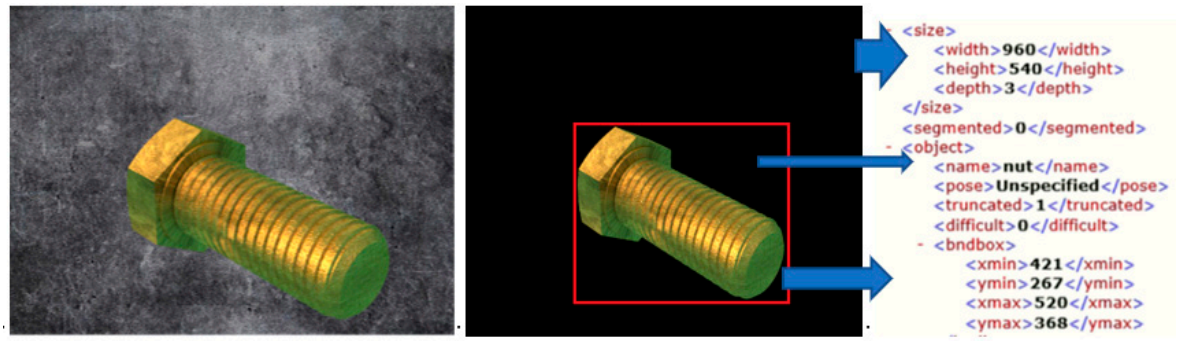


Figure 4. The generated 2D image from Blender software with Python script (**left**), red rectangle window of the part position generated by OpenCV (**middle**), image parameters in the XML structure (**right**).

3.3. Training Process

Two CNN models were selected for experiments. The first model (Faster RCNN Inception v2) was selected because of its high accuracy, but with the disadvantage of its network size and recognition speed. The second selected model, Mobilenet V2, is more suitable for embedded devices, because it is optimized for low performance computing devices, e.g., smart glasses with Android OS.

They are capable of processing limited input image sizes only due to their optimization for lower resolution (usually up to 300 × 300 pixels). Such a low resolution provides unreliable results in the detection of multiple objects (nut, screw, washer) in one image because of insufficient object detail, for example, screw threads or the edges of hexagonal nuts.

The inception model has a size of 52 MB as opposed to MobileNet V2, which has a size of only about 19 MB. For the training process, we did not need to teach from scratch. We used a trained model with a general dataset, coco, with almost the same results. This technique only removes the last layers, which are then required to be taught again by transfer learning.

Teaching the CNN model from scratch is not necessary, because the weights of neurons on lower hidden layers of the CNN model only store general information about images. The CNN can be trained on a very different set of general object types (peoples, animal, common household objects), and after transfer learning, it can recognize very specific objects, like the industrial parts described in this paper.

Generally, the CNN network must be pretrained with very large sets of samples of recognized objects, which can take several weeks. The transfer learning process can reduce this training time to one day or several hours, for example, by using an advanced graphics card.

For the Inception V2 model, we performed transform learning with 200,000 cycles with an achieved mAP (mean average precision) of 1.8×10^{-3} for localization and 1.37×10^{-3} for classification, and the transform learning process took 10 h and 33 min. For the Mobilenet V2 model, we reached an mAP of 0.5184 for classification and 0.03830 for localization after 18,000 cycles and the training time was 7 h and 44 min.

The teaching procedure trains the CNN model in parallel for classification and localization with outputs: X, Y position, type of object, classification probability (Figure 5).

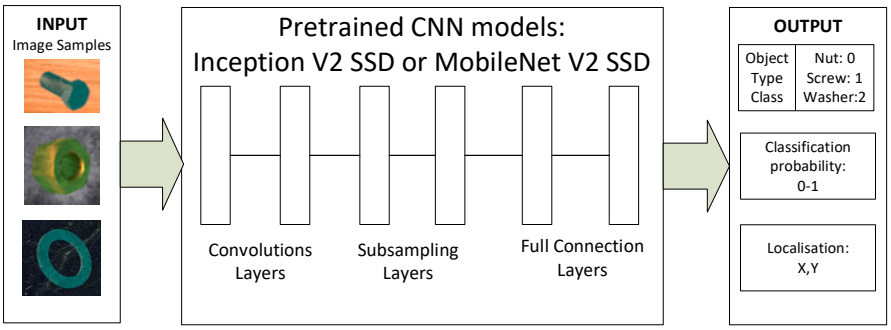
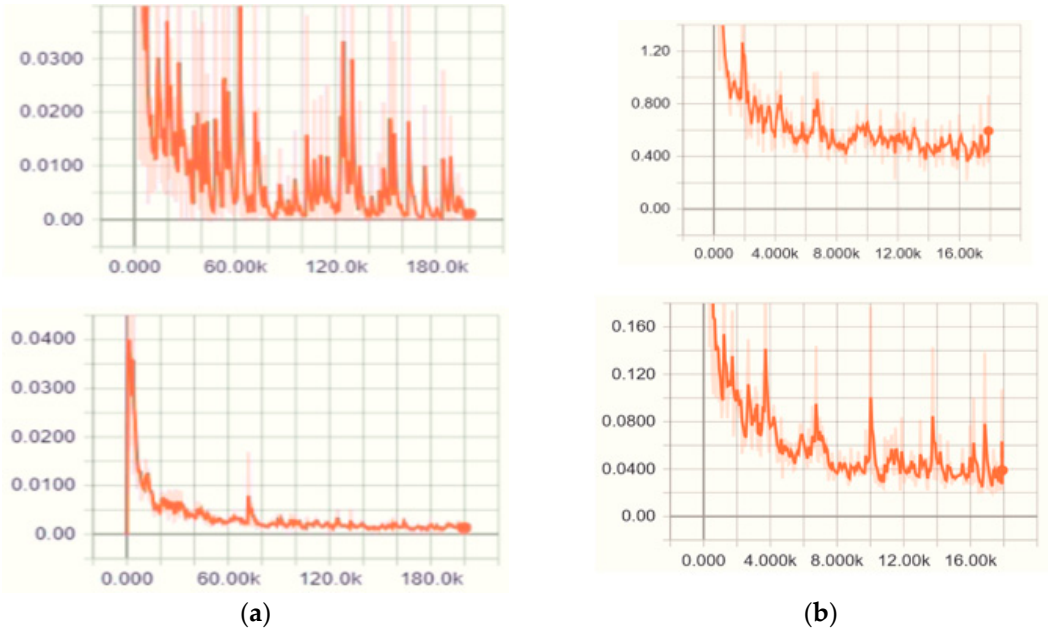


Figure 5. The principle of the use of pretrained CNN models for object recognition.

We used an integrated web interface to monitor the training process in graphical form by the TensorFlow library. Separate graphs of the transfer learning process for classification and localization are shown in Figure 6, where the unit on the x-axis is the number of cycles and the unit on the y-axis is mAP.



(**a**) (**b**)

Figure 6. The training process for the classification of objects (top graphs) and their position (bottom graphs) for: (**a**) the CNN model, Inception V2; (**b**) the CNN model, MobileNet V2.

The proposed automated process for the generation of 2D samples from a 3D virtual model based on a script in Python was extended to a web user interface to allow for a simple setup and management by a user friendly graphical interface. An example of automated input sample generation and training using a web interface is shown in Figure 7.

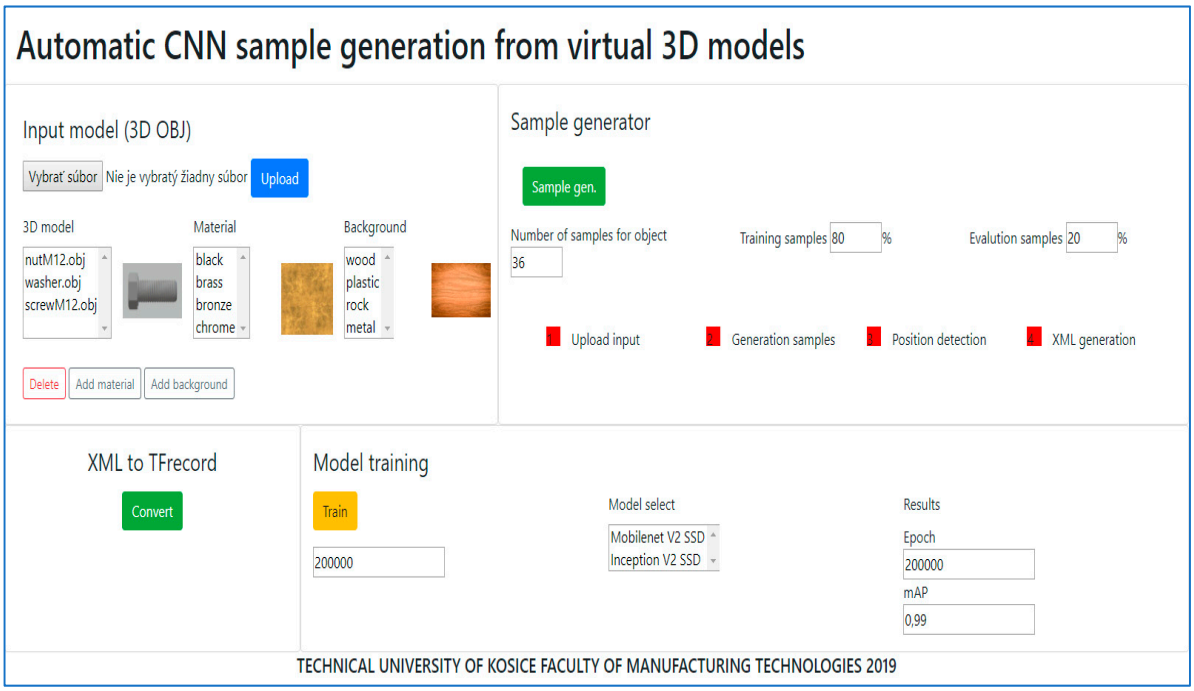


Figure 7. An example of a remote web interface for the execution of 2D sample generation and CNN training.

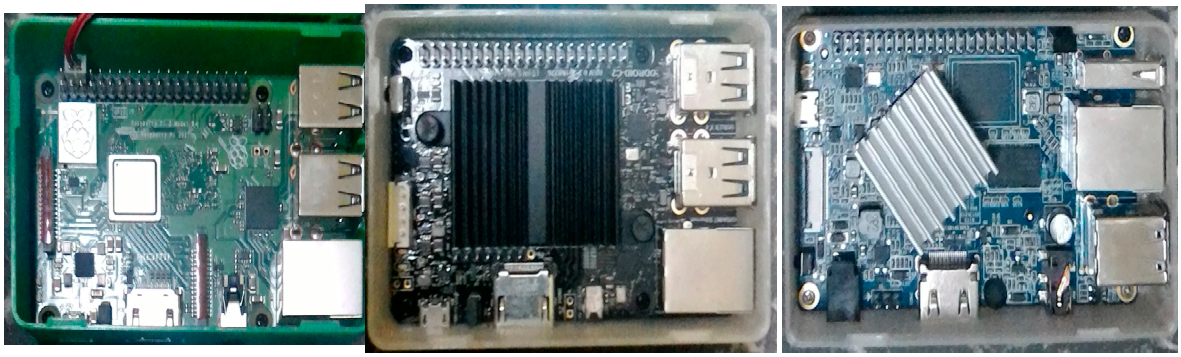
4. Experiments

The experiments consisted of object classification and their localization, measurement of the recognition processing delay, and, finally, a reliability evaluation of the recognition process using the PC (Personal Computer) platform, embedded systems, and VR/AR devices.

4.1. Experiment Environment

The transform learning process was tested in the TensorFlow framework using the scripting language, Python, and the Anaconda environment was used to separate both CNN teaching models. The experimental transfer learning was sped up by CUDA cores of a NVIDIA GTX1060 graphics card. The first evaluation of the results was tested in Python on the PC platform.

It is suitable to prepare experiments on standard development boards before the real implementation of the recognition solution to specialized devices for virtual reality (Cardboard VR) or augmented reality (smart glasses AR). We chose the following embedded systems as mobile testing platforms: Raspberry PI 3 B+, ODROID C2, and Orange PI PC Plus. The most suitable open source library for CNN model testing is teh OpenCV library (C ++ version). The selected platforms are shown in Figure 8.



(**a**) (**b**) (**c**)

Figure 8. Tested embedded platforms: (**a**) Raspberry PI 3B+ [1.4 GHz]; (**b**) ODROID C2 [1.5 GHz]; (**c**) Orange PI PC Plus [1.6 GHz].

Figure 9 shows the devices used for testing the training models in augmented and virtual reality devices.

(**a**) (**b**)

Figure 9. The used devices in the experimental testing: (**a**) Cardboard VR device with smartphone; (**b**) smart glasses, Epson Moverio BT350.

Our development focuses on using it in the production process, in which augmented reality can help to improve the collaborative assembly process. Thus, for the assisted assembly, we need to transform the trained models from a PC to a wearable device, for example, to Android OS included in the VR/AR device. For that reason, the development of the software for the combination of the unity environment for visualization and OpenCV C++ for CNN model processing was needed.

Software implementation required the combination of the three software products in one collaboration unit during development:

- TensorFlow (Python)—teaching CNN (model export and optimization).
- Android Studio (Java/C++)—CNN execution (recognition processing delay minimization).
- Unity (C#)—3D engine—data visualization (3D model visualization in augmented reality).

The deployment principle to the software and hardware is shown in Figure 10.

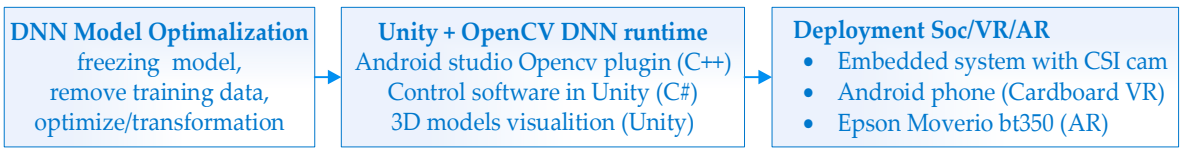


Figure 10. The deployment solution for testing the AR/VR devices.

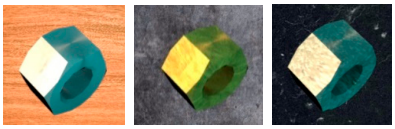
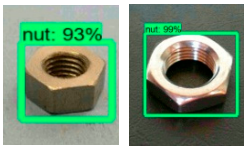
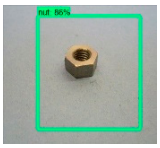
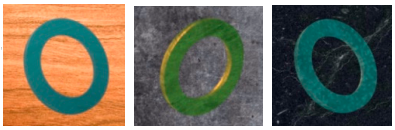
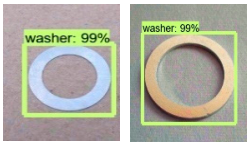
Android Studio IDE was used to program the unique DLL plugin for fast CNN model execution written in the C++ language as a wrapper for the OpenCV library. This DLL is developed and

maintained in our Department and provides a bridge to the unity engine. The plugin is currently compiled for x64, armv8, and armv7 SoC (System on Chip). The unity framework provides a user-friendly interface across several platforms to create an assisted assembly 3D scene compatible with Android, Linux, and Windows. Unity control logic is written in the C# language and provides image input and output from the DLL plugin. We improved the performance of the realized application by translating C# to C++ by IL2CPP compiler inside of the unity build system.

4.2. Evaluation Measure

The proposed idea of using virtual samples was first tested with different samples at randomly selected angles and backgrounds to verify that the learned model is independent of the material/background. It was confirmed by the initial experiments that the recognition of virtual samples by trained CNN models can work universally and in any condition. The next experiments were provided with the real images of industrial parts as shown in Table 1.

Table 1. Verification of the reliability for trained CNN models by real samples.

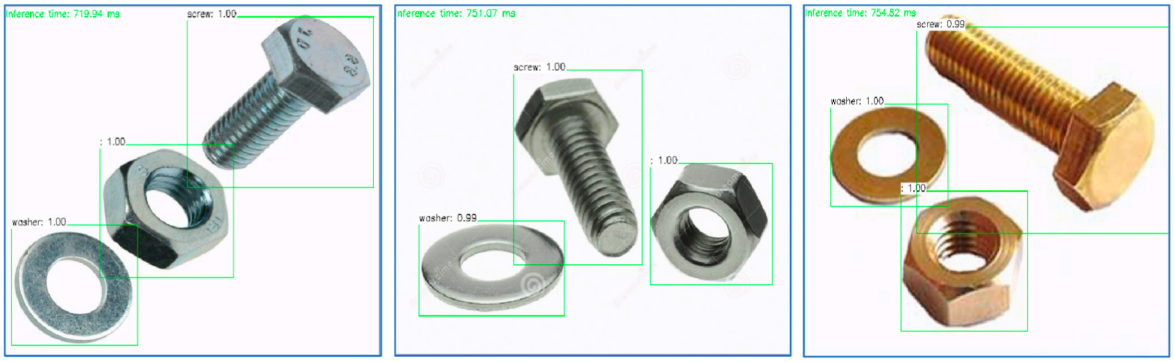
Part Name	Training: Example of Virtual Part Samples (Changed Material/Background)	Testing with TensorFlow Framework (Python Execution Experiments)		mAP min/max
		Inception V2	Mobilenet V2	
Screw M12				0.96/0.99
Nut M12				0.93/0.86
Washer 12				0.99/1

Although the difference in the precision of the classification between Inception V2 and Mobilenet V2 is not significant, Inception is much more precise in position identification. However, if real-time hardware implementation is considered, the execution time of the recognition is crucial. If a low cost and mobile recognition device is needed, this solution must be implemented in the embedded platform.

4.3. Experiment Process

The assisted assembly process requires that the position of a detected part is known because more recognized parts can be placed in the field of view (FOV), so multiple detection in one image has to be used. The teaching process with single shot detection (SSD) for part location can solve this issue. The use of a white background in the recognition experiments with real samples was possible because this background was not used for the training and evaluation process of the CNN models. The results of the testing of the recognition precision and recognition processing delays (more information about the research on the delayed scaled consensus problems can be found in [27]) were acquired by a remote desktop and command line SSH shell from the embedded devices.

The experiments were carried out with multiple detection in a real image as shown in Figure 11.

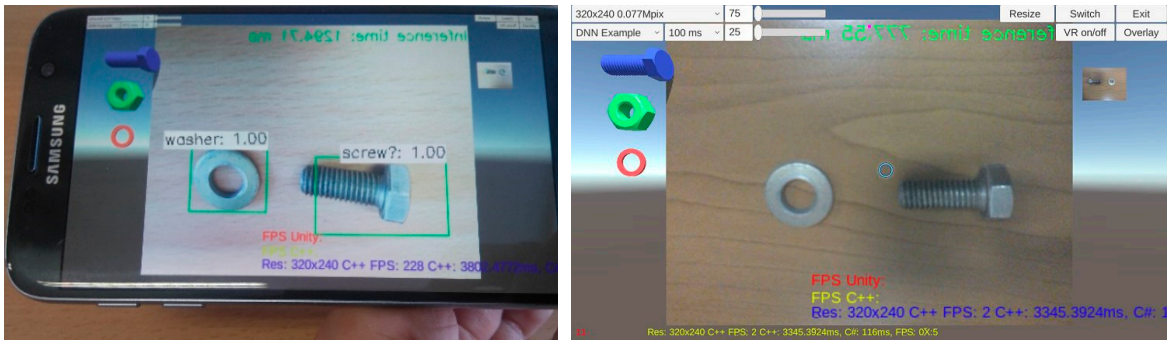


(**a**) (**b**) (**c**)

Figure 11. Multiple recognition for all parts in one image executed by the CNN model, Inception V2 SSD. (**a**–**c**) are three different parts layouts.

The results for multiple recognition were very precise; we achieved an mAP from 99% to 100%. More reliable results of object classification and localization can be achieved by the Inception V2 advanced CNN model, but with a higher recognition processing delay. Thus, this model is not suitable for AR/VR devices. The MobileNet V2 reduced CNN model is faster (lower processing delay), but the recognition results are less reliable due to limited image resolution in the training process.

An example of nut/screw recognition with the implemented MobileNet V2 CNN model using a smartphone and smart glasses is shown in Figure 12.



(**a**) (**b**)

Figure 12. An example of nut/screw recognition with the MobileNet V2 CNN model by: (**a**) Smartphone Samsung S7; (**b**) smart glasses Epson Moverio BT350 (screenshot).

4.4. Experiment Results

The results of the experiments with MobileNet V2 SSD (inference time, position detection, and classification) are shown in Figure 13.

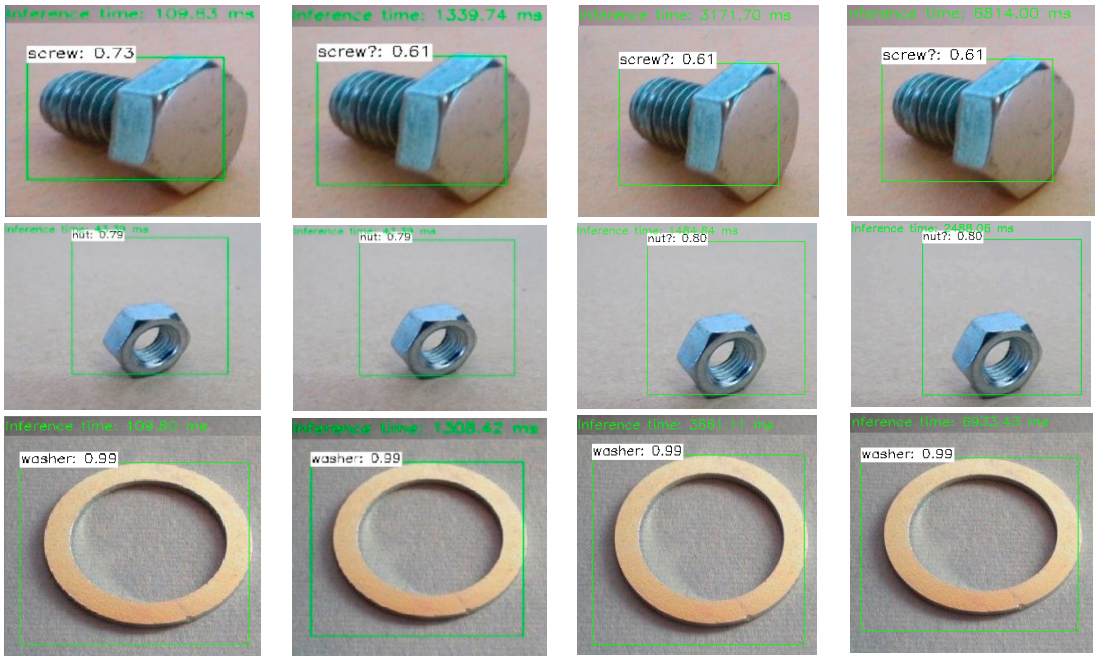


Figure 13. CNN recognition processing delays Mobilenet V2: (**a**) PC—Intel Core i5-8400 (2.8 GHz) Windows 10 [x64]; (**b**) Odroid C2—Amlogic ARM A57 (1.5 GHz) Armbian [ARMv8]; (**c**) Orange PI PC Plus—Allwinner ARM A7 (1.6 GHz) Armbian [ARMv7]; (**d**) Raspberry PI 3—Broadcom ARM A53 (1.4 GHz) Armbian [ARMv8].

The results of the experiments with Inception V2 SSD (inference time, position detection, and classification) are shown in Figure 14.

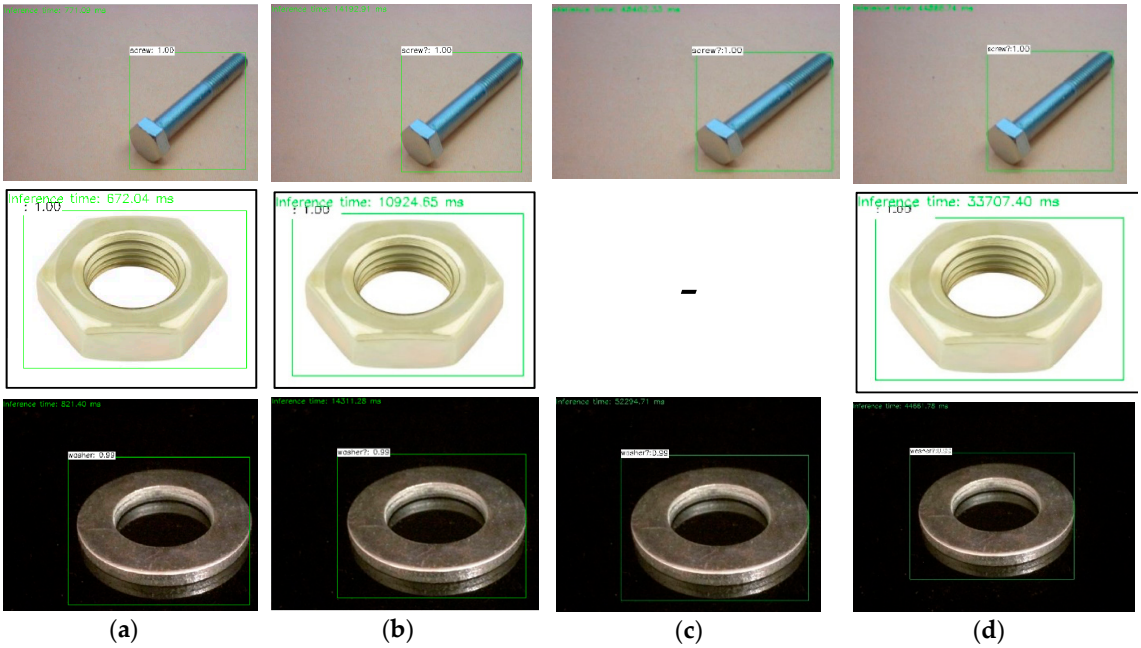


Figure 14. CNN recognition processing delays Inception V2: (**a**) PC—Intel Core i5-8400 (2.8 Ghz) Windows 10 [x64]; (**b**) Odroid C2—Amlogic ARM A57 (1.5 GHz) Armbian [ARMv8]; (**c**) Orange PI PC Plus—Allwinner ARM A7 (1.6 GHz) Armbian [ARMv7]; (**d**) Raspberry PI 3—Broadcom ARM A53 (1.4 GHz) Armbian [ARMv8].

The experiments with the CNN model recognition processing delays, classification, and position detection using AR/VR devices are shown in Figure 15.

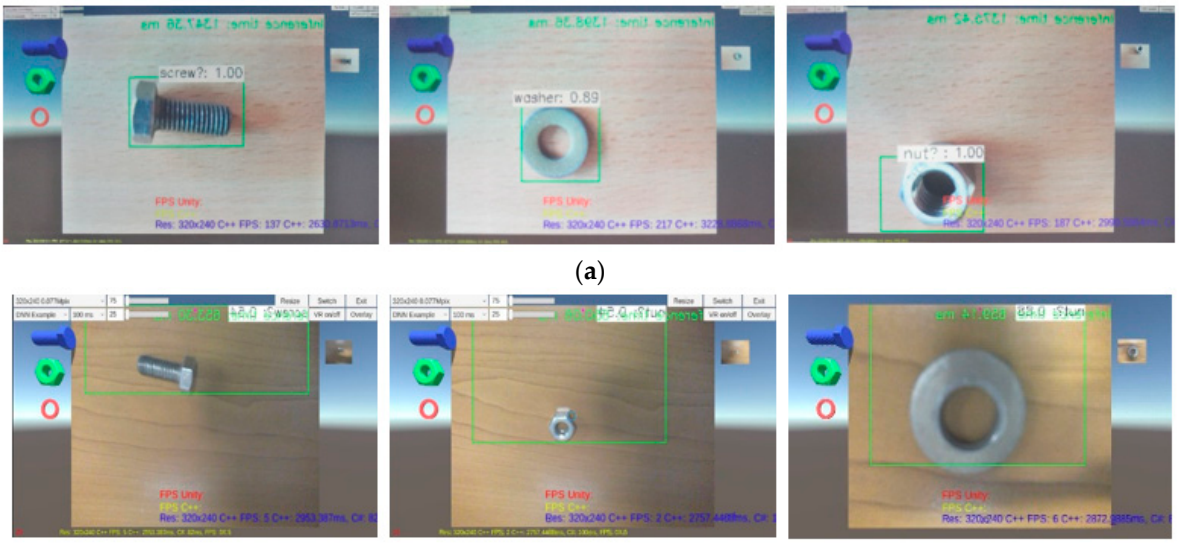


Figure 15. The experiments with AR/VR devices' CNN performance of Mobilenet V2: (**a**) Cardboard VR device with mobile phone Samsung S7; (**b**) AR device smart glasses Epson Moverio BT350.

A comparison of the cumulated results of CNN recognition processing delays for different processing platforms is shown in Table 2.

Table 2. Comparison table of CNN recognition processing delays for the desktop platforms, Windows x86/Linux ARM.

Processor Type\Recognition Processing Delays [ms]	Screw (700 × 525) Inception/Mobilenet	Nut (390 × 300) Inception/Mobilenet	Washer (700 × 525) Inception/Mobilenet
Intel Core i5-8400 2.8 GHz (HP Omen)	771.09/109.83	672.04/43.39	821.40/106.18
Amlogic A57 1.5 GHz (Odroid C2)	14,192.91/1339.74	10,924.65/675.78	14,311.28/1308.42
Allwinner H3 A7 1.6 GHz (Orange Pi Lite Plus)	48,482.33/3171.07	-/1484.84	52,294.71/3661.11
Broadcom A53 1.4 GHz (Raspberry Pi 3)	44,886.64/6814.01	33,707.40/2488.06	44,661.78/6933.43

The cumulated results of the measured recognition processing delays with VR/AR devices are shown in Table 3.

Table 3. Comparison table of CNN recognition processing delays for a screw, nut, and washer running on Android OS (AR/VR).

Device Type\Delay [ms]	Screw (320 × 240)	Nut (320 × 240)	Washer (320 × 240)
Cardboard VR Exynos A53 4 × 2.3 GHz Android 8.1 (Samsung S7)	1347.36	1375.42	1389.36
Smart Glasses AR Atom x5 4 × 1.44 GHz Android 5.1 (Epson Moverio bt-350)	653.30	650.06	634.48

AR/VR devices were tested only with the Mobilenet V2 CNN model. Embedded SoC devices with Linux OS tested in the previous subchapter provided a similar performance for the execution of the CNN model. Smart glasses with Android (Intel Atom), Epson Moverio BT350, can reach a recognition processing delay of about 650 ms and the Android mobile phone (ARM) for Cardboard

VR was about 1400 ms. In summary, AR/VR devices equipped with embedded ARM or Intel Atom processors can reach about 0.6 to 1.4 FPS only, which is suitable if the operator does not move very fast during the assembly process. For comparison, we also executed the CNN model in a standard PC with an Intel i5-8400 processor, in which a recognition processing delay of 43 ms was achieved (~20 fps).

4.5. Discussion

Continuous object detection is necessary for assisted assembly because virtual (augmented) reality devices are now worn by assembly staff, thus they can operate in constant movement. Devices controlled by a standard PC with an Intel x 86 processor can achieve a reasonable framerate of 20 fps with a reduced resolution of about 390 × 300 pixels, which is enough for fluent identification in an automated line with a conveyor.

The tested embedded systems can reach only 0.5 to 1 fps with Mobilenet V2, which is usable for assisted assembly of a fixed position of the main part. Experiments with Inception V2 combined with embedded platforms showed that it is unusable because recognition processing delays are very high at about 10 and more seconds.

An application of augmented reality in assisted assembly requires multiple detections of assembly parts with overlays. However, there is a problem with the input dataset for the training of recognition models because the preparation of samples that partly overlay is a very complicated process. So, this requires automated input data preparation using, for example, 3D CAD software. Our new idea for the generation of realistic 3D model parts' overlay is the implementation of Newton dynamics with the free fall of parts to a fixed surface. This principle can create overlays with all the tested parts without collision.

The automated sample generation is simplified through the use of a designed user-friendly web interface. The user can upload the CAD model from the 3D design software and choose the object's material and background. A CNN model can be remotely trained in the cloud and the results can be published as a downloadable content to any platform (desktop OS, Android).

The next problem is the part overlay in assembly. Our idea to solve this task is using the 3D construction software feature for the assembly explode and to prepare many variants of the finished product. However, the Blender software does not support this feature so another scripting language from 3D design software must be used.

Further experimental work will focus on learning and testing other pretrained models, mainly quantized CNN models, by the same methodology with virtual 3D models, which are much faster on an Android mobile platform.

5. Conclusions

It is clear from the experimental results with the 3D virtual model training technique that CNN models can be learned without real frames. The experiments were carried out using two CNN models: Inception V2 SSD and Mobilenet V2 SSD. For both CNN models, we achieved probabilities ranging from 0.93 to 0.99 in individual recognition and from 0.91 to 0.99 in multiple object recognition. The obtained results show that the use of virtual models for the teaching of CNN is a potential method for future implementation.

Moreover, as a part of the experimentation, we used two different real-hardware implementations, one in a VR device and the second one in an AR device. The device based on Cardboard VR with an external high-end mobile phone provided a much better performance than the AR embedded processing unit with transparent display, but the need for image processing in VR glasses requires higher power consumption. We achieved only about 1.8 to 2 fps at a very small resolution of 320 × 240 pixels. Using the Inception V2 SSD model, precise object recognition was achieved, but with a very long recognition processing delay (10 s), which makes it unsuitable for wearable devices. The optimal performance of the object recognition speed (20–30 fps) is currently possible only by a

PC platform with an external graphics card, Nvidia 10 × 0 or AMD Vega series, for example, the VR headset HTC Vive Pro.

A very interesting part for future experiments is a comparison of the execution time (recognition processing delay) of the same CNN model in a different CNN framework: TensorFlow, Caffe1/2, Torch, Microsoft Cognitive Toolkit, MXNet, etc. The next speed increase can be realized by the transfer model from the basic embedded platform to a specific board with a CUDA graphical processing unit (GPU), for example, NVIDIA Tegra TX1 or Android hardware with APU (e.g., Huawei P20 Pro mobile phone used in Cardboard VR).

Author Contributions: Methodology, K.Z.; software, K.Z.; validation, K.Z., A.H. and J.P.; formal analysis, A.H.; investigation, A.H.; resources, J.P.; data curation, P.L.; writing—original draft preparation, K.Z.; writing—review and editing, J.P.; visualization, P.L.; supervision, J.P.; project administration, J.P.; funding acquisition, J.P.

Funding: This work was supported by the Slovak Research and Development Agency under the contract No. APVV-15-0602 and also by the Project of the Structural Funds of the EU, ITMS code: 26220220103.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Pavlenko, I.; Simonovskiy, V.; Ivanov, V.; Zajac, J.; Pitel, J. Application of artificial neural network for identification of bearing stiffness characteristics in rotor dynamics analysis. In *Advances in Design, Simulation and Manufacturing*; Lecture Notes in Mechanical Engineering; Springer: Cham, Switzerland, 2019; pp. 325–335.
2. Zidek, K.; Piteľ, J.; Hošovský, A. Machine learning algorithms implementation into embedded systems with web application user interface. In Proceedings of the IEEE 21st International Conference on Intelligent Engineering Systems (INES 2017), Larnaca, Cyprus, 20–23 October 2017; pp. 77–81.
3. Zidek, K.; Maxim, V.; Pitel, J.; Hosovsky, A. Embedded vision equipment of industrial robot for inline detection of product errors by clustering-classification algorithms. *Int. J. Adv. Robot. Syst.* **2016**, *13*. [CrossRef]
4. Shang, Y. Deffuant model of opinion formation in one-dimensional multiplex networks. *J. Phys. A Math. Theor.* **2015**, *48*, 395101. [CrossRef]
5. Pavlenko, I.; Trojanowska, J.; Ivanov, V.; Liaposhchenko, O. Scientific and methodological approach for the identification of mathematical models of mechanical systems by using artificial neural networks. In *Innovation, Engineering and Entrepreneurship*; Lecture Notes in Electrical Engineering; Springer: Cham, Switzerland, 2019; pp. 299–306.
6. Židek, K.; Maxim, V.; Sadecký, R. Diagnostics of errors at component surface by vision recognition in production systems. *Appl. Mech. Mater.* **2014**, *616*, 227–235. [CrossRef]
7. Židek, K.; Hošovský, A.; Dubjak, J. Diagnostics of surface errors by embedded vision system and its classification by machine learning algorithms. *Key Eng. Mater.* **2016**, *669*, 459–466. [CrossRef]
8. Židek, K.; Hosovsky, A.; Piteľ, J.; Bednár, S. Recognition of assembly parts by convolutional neural networks. In *Advances in Manufacturing Engineering and Materials*; Lecture Notes in Mechanical Engineering; Springer: Cham, Switzerland, 2019; pp. 281–289.
9. Sarkar, K.; Varanasi, K.; Stricker, D. Trained 3D models for CNN based object recognition. In Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (Visigrapp 2017), Porto, Portugal, 27 February–1 March 2017; pp. 130–137.
10. Socher, R.; Huval, B.; Bhat, B.; Manning, C.D.; Ng, A.Y. Convolutional-recursive deep learning for 3D object classification. In Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12), Lake Tahoe, NV, USA, 3–6 December 2012; pp. 656–664.
11. Su, H.; Qi, C.R.; Li, Y.; Guibas, L.J. Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV'15), Santiago, Chile, 7–13 December 2015; pp. 2686–2694.
12. Tian, Y.; Li, X.; Wang, K.; Wang, F.Y. Training and testing object detectors with virtual images. *IEEE/CAA J. Autom. Sin.* **2018**, *5*, 539–546. [CrossRef]

13. Nie, W.Z.; Cao, Q.; Liu, A.A.; Su, Y.T. Convolutional deep learning for 3D object retrieval. *Multimed. Syst.* **2017**, *23*, 325–332. [CrossRef]
14. Peng, X.C.; Sun, B.C.; Ali, K.; Saenko, K. Learning deep object detectors from 3D models. In Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV'15), Santiago, Chile, 7–13 December 2015; pp. 1278–1286.
15. Cucos, M.M.; Pista, I.M.; Ripanu, M.I. Product engineering design enhancing by parameterizing the 3D solid model. *MATEC Web Conf.* **2018**, *178*, 05011. [CrossRef]
16. Nguyen, C.H.P.; Choi, Y. Triangular mesh and boundary representation Combined Approach for 3D CAD Lightweight Representation for Collaborative Product Development. *J. Comput. Inf. Sci. Eng.* **2018**, *19*, 011009. [CrossRef]
17. Pollák, M.; Baron, P.; Zajac, J. Automatization process of designing the technological documentation by tools of comprehensive CAD-CAM-CAE system. *MM Sci. J.* **2016**, 942–946. [CrossRef]
18. Pollák, M.; Baron, P.; Telišková, M.; Kočiško, M.; Török, J. Design of the web interface to manage automatically generated production documentation. *Tech. Technol. Educ. Manag. TTEM* **2018**, *7*, 703–707.
19. Gopalakrishnan, K. Deep Learning in data-driven pavement image analysis and automated distress detection: A review. *Data* **2018**, *3*, 28. [CrossRef]
20. Mao, K.; Lu, D.; E, D.; Tan, Z. A case study on attribute recognition of heated metal mark image using deep convolutional neural networks. *Sensors* **2018**, *18*, 1871. [CrossRef] [PubMed]
21. Pham, G.; Lee, S.-H.; Kwon, O.-H.; Kwon, K.-R. Anti-3D weapon model detection for safe 3D printing based on convolutional neural networks and D2 shape distribution. *Symmetry* **2018**, *10*, 90. [CrossRef]
22. Liu, H.; Wang, L. An AR-based worker support system for human-robot collaboration. *Procedia Manuf.* **2017**, *11*, 22–30. [CrossRef]
23. Gerstweiler, G.; Furlan, L.; Timofeev, M.; Kaufmann, H. Extraction of structural and semantic data from 2D floor plans for interactive and immersive VR real estate exploration. *Technologies* **2018**, *6*, 101. [CrossRef]
24. Gao, Q.H.; Wan, T.R.; Tang, W.; Chen, L.; Zhang, K.B. An improved augmented reality registration method based on Visual SLAM. In *E-Learning and Games*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2017; pp. 11–19.
25. Židek, K.; Hošovský, A. Wireless device based on MEMS sensors and bluetooth low energy (LE/smart) technology for diagnostics of mechatronic systems. *Appl. Mech. Mater.* **2014**, *460*, 13–21. [CrossRef]
26. Hruboš, M.; Svetlík, J.; Nikitin, Y.; Pirník, R.; Nemec, D.; Šimák, V.; Janota, A.; Hrbček, J.; Gregor, M. Searching for collisions between mobile robot and environment. *Int. J. Adv. Robot. Syst.* **2016**, *13*. [CrossRef]
27. Shang, Y. On the delayed scaled consensus problems. *Appl. Sci.* **2017**, *7*, 713. [CrossRef]