

Design and Performance Analysis of Docker-Based Smart Manufacturing Platform Based on Deep Learning Model

Soonsung Hwang, Jaehyoung Lee, Dongyeon Kim, and Jongpil Jeong(✉)

Department of Smart Factory Convergence, Sungkyunkwan University, Suwon, Gyeonggi-do 16419, Republic of Korea
{suns303,objective,kdy5958,jpjeong}@skku.edu

Abstract. Breakdown of equipment causes very large damage to the factory. Research is continuously being conducted to prevent break down of equipment by detecting abnormal signs before equipment failure. This paper proposes an anomaly detection for system architecture based on a docker container. A docker is a virtualized container with many performance and scalability advantages. We have used the deep learning model of Autoencoder to effectively anomaly detection and its performance has been proven through experiments.

Keywords: Anomaly detection · Autoencoder · Docker · Smart Manufacturing · RMSE

1 Introduction

Docker, a container-based virtualization technology, provides an environment in which all containers can run applications on the same system. Developers can easily customize and package their runtime environment from a set of files called Docker Images. Once the Docker Image is created, the user can run the image on any host computer using the Docker Engine. The ability to build a single image of the docker and run it from anywhere greatly simplifies the development, delivery, and deployment of applications [1]. More and more companies are now building container-based platforms using Docker. The docker is very easy to develop and extensible [2].

We propose a system architecture for manufacturing using a docker. We demonstrate the process of anomaly detection in equipment through system architecture [3,4]. In this paper is based on case studies applied to actual factories. Our manufacturing plant is a PCB gold plating process. If equipment failure occurs breakdown during production, this will lead to a large loss of the plant. Equipment can be divided into normal section, abnormal symptom section, breakdown. Before the breakdown, abnormality symptom interval always occurs. Unlike the normal section, there are several different symptoms in the

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11624, pp. 94–104, 2019.
https://doi.org/10.1007/978-3-030-24311-1_7

abnormal symptom section. If an abnormal symptom is caught in the abnormal symptom section and the action is taken, the equipment can be prevented from breakdown. We aim to anomaly detection in the anomaly section before equipment failure [5]. We propose system architecture that From the sensor data collection stage attached to the facility, to the process of alarming the operator after anomaly detection using artificial intelligence. We use Autoencoder model for anomaly detection [6,7].

The composition of our paper is as follows. In Sect. 2, related work describe the Docker platform and the Autoencoder model. In Sect. 3, we proposed system architecture. Section 4 shows the experiment results. Finally, our conclusion is drawn in Sect. 5.

2 Relate Work

2.1 Docker-Based Platform

Docker is the world's leading software container platform. Containers are a single object that contains all the necessary components for the software to run, and each container operates in an isolated environment. Therefore, containerized software runs the same way regardless of the operating system environment. This looks similar to a virtual machine, but unlike a VM, the Docker virtualizes the operating system, not the hardware. Thus, the container can share the OS kernel with other containers, and each container runs as an isolated process in user space. Therefore, containers are more portable and efficient than VMs.

However, containers are not necessarily better than VMs. First, containers can be more vulnerable to security. If a user has superuser privileges in a container, theoretically, that user may have problems with the superuser's privileges to the underlying OS underneath the container. It is therefore recommended that containers be handled in the same way as other server applications. Containers also make it easy for developers to develop applications because they make the application-driven environment a safe that does not need to be opened, but conversely developers are more difficult to understand the contents of the safe. This means you can easily distribute your app, but your app will be more likely to be deployed on the wrong container. Finally, it's a good idea to divide the system to be distributed into smaller parts, but on the contrary, there are more parts to manage. For the above reasons, Docker supports VM, so container and VM can coexist and it is recommended to use containers and VM appropriately according to the situation. If you do not know when and how you will use it, it is better to use the VM in situations where you need to run a container on a system that needs to run multiple copies of the same application and run multiple applications on a single system.

If you use a docker in the right environment, the docker makes it easier to build a virtualized environment based on the container. The docker consists of an image and a container, and the set value can be composed into an image and executed in a container. Compared to virtual machines, they run at a much

faster speed and do not virtualize hardware, which improves memory access and file system performance.

2.2 Autoencoder Model

We use Autoencoders as anomalous detection models. An Autoencoder is a neural network that copies inputs to outputs [8]. Figure 1 shows the structure of Autoencoder. The Autoencoder consists of an encoder and a decoder. The encoder is called the recognition network and translates the input into an internal representation. The decoder is called generative network and converts its internal representation to output [9]. Because the auto-encoder reconstructs the input, the output is called reconstruction, and the loss function is computed with the difference between input and reconstruction [10].

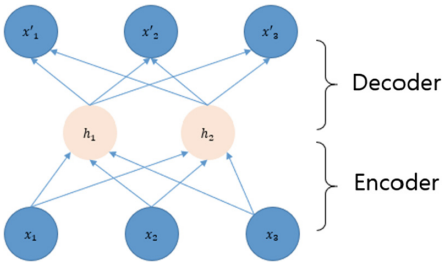

Fig. 1. Autoencoder model architecture.

The Autoencoder can only be compressed with data similar to the trained data. In addition, Autoencoder automatically learns from data, which is a useful property. Learning from data automatically means that you can easily train a particular type of algorithm that works well for certain types of input values. It is only necessary to properly train the data without the need for new engineering measures. Encoders and decoders are chosen as parametric functions and differentiated in terms of distance functions, so the parameters of the encoding/decoding function can be optimized using a stochastic gradient descent to minimize reconstruction loss.

The formula of Autoencoder is as follows.

$$Encoder : h = f'_\theta(x) = s(Wx + b') \tag{1}$$

$$Decoder : x' = g'_\theta(h) = s(W'h + b') \tag{2}$$

In 2012, a way to apply Autoencoder has been found in greedy layer-wise pretraining of deep convolutional neural networks [11]. However, as the random weight initialization schemes were found to be sufficient to train deep networks from scratch, the Autoencoder quickly disappeared from fashion. In 2014, batch normalization [12] began to allow much deeper networks, and by the end of 2015 it was possible to arbitrarily train deep networks using residual learning [13].

3 Docker-Based Smart Manufacturing Platform Based on Deep Learning Model

We propose a Docker-based Smart Manufacturing Platform based on Deep Learning Model as shown in Fig. 2. Our platform runs DAQ, Database, Kafka, Web application, Predictor, and Model Training containers on the docker engine. The docker-based platform provided function that Application Scheduling, Container Scheduling, Service Discovery, Container Cluster Management, and Container Networking.

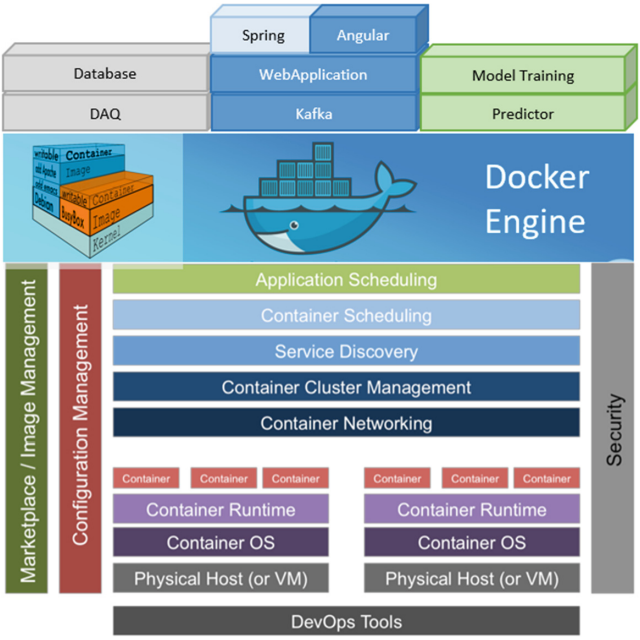

Fig. 2. Docker-based Smart Manufacturing Platform.

This section describes the architecture of the Autoencoder based anomaly detection and monitoring system proposed in this paper. As shown in Fig. 3, the structure of our system is as follows.

Step 1: Attach vibration sensor from equipment and collect data.
Step 2: Send PLC data and sensor data of the equipment collected from DAQ to database.
Step 3: Send the data stored in the database to create a learning model.
Step 4: Generate normal data learning model using auto-encoder model.
Step 5: Create a normal data learning model and load the learning model into edge computing.

Step 6: Send the data collected by DAQ to Predictor in real time.
Step 7: Normal data When the normal data is entered into the learning model, the RMSE becomes low, and when the untrained data enters the data, the RMSE becomes high.
Step 8: Monitor the RMSE value in real time and pay attention to whether abnormalities occur periodically.
Step 9: Periodically, if the RMSE value is high, it is recognized as an abnormal symptom and an alarm is sent to the user.

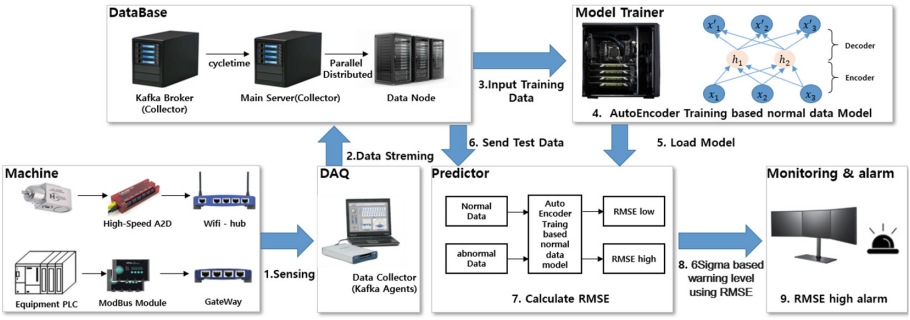

Fig. 3. System architecture.

3.1 Machine

This component is part of the factory facility. This is the part where the sensor is attached to the part where the measurement is desired in the facility and the corresponding data is collected. Here, you have to be careful about what data to attach to where on the installation. In this paper, a vibration sensor is attached to the crane of the facility and the motor and the bearing to collect data. The crane moves horizontal, vertical, and pitch. Horizontal has 3 sensors, Vertical has 2, Peach has 1 sensor, and 6 sensors per crane.

3.2 DAQ: Data Acquisition

This component is the part that collects facility data. We collect sensor data and PLC data from the machine and stream data using Apache Kafka which is Hadoop ecosystem. Apache Kafka is a distributed messaging system that outperforms traditional messaging systems through an architectural design that specializes in high-volume real-time log processing. It is based on the Pub-Sub model and consists of Producer, Consumer, and Broker. The Broker manages the message based on the topic, and the Producer generates a message for the topic and forwards the message to the Broker. When a Broker categorizes and

stacks messages received by a topic, Consumers who subscribe to the topic process the message. The difference with existing messaging systems is that they are designed based on distributed systems, which makes it easier to configure distribution and replication over existing messaging systems. In addition, TCP - based protocol with simple message header reduces the overhead, and it can reduce the number of TCP/IP roundtrips by sending a large number of messages to the broker at once.

Figure 4 is the process of data collection process used in this paper. Figure 4 shows Sensor Data Collector and PLC Event Collector. The sensor data in this paper is collected by attaching a vibration sensor to the motor and the bearing to collect the vibration data generated during the movement of the equipment. The sensor data collects 2096 data per 0.5 s. The PLC event data is data indicating the start and end of the facility. The Sensor Data Collector collects data from the process, and the PLC Event Collector collects events from the start and end of the PLC. These two collectors are collected by SensorData Collector through Kafka Broker, stored in parallel by HBase, periodically send data to Kafka Broker in PLC Event Collector, and receive data by PLC Data Collector.

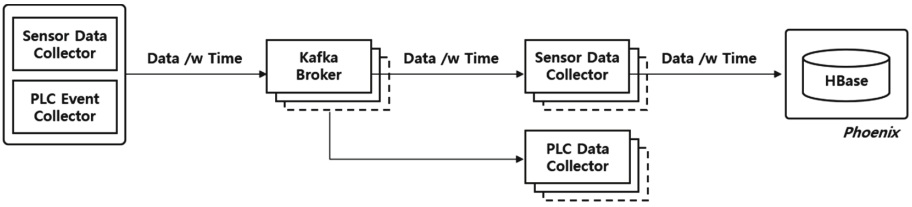

Fig. 4. Data gathering process.

3.3 DataBase

This component is a database. Our system uses two kinds of database. First, the sensor data received from the DAQ component is stored in parallel to the Hadoop ecosystem NoSQL Hbase. We use NoSQL because we have a lot of sensor data. The second database uses postgres, an RDB that simply queries the data results in a web application. The reason for using two databases is that it is efficient to save the data that does not modify the value like sensor data in Nosql and to build the system by saving the data that needs CRUD in RDB to the web application.

3.4 Model Trainer

This component generates a normal data learning model using an Autoencoder model. Figure 5 shows the Trainer process of the model. We created a learning model using the Python library Tensor flow using the PLC data collector

from the DAQ component and the sensor data from HBase of the DataBase component. We train only normal data with Autoencoder model.

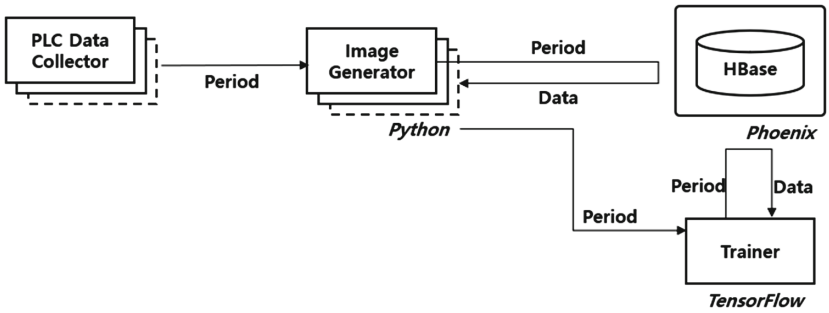

Fig. 5. Model training process.

3.5 Predictor

This component predicts by loading the learned model. There are two ways to actually predict the learned model. The first is to use a file saved as a checkpoint. Checkpoints only store the variables used when creating the model. To load the data, you have to draw the graph with the tensorflow code and load it. The second way is to store the graphs together rather than just storing them like checkpoints. It is stored in Protocol Buffer type. This file is also called Pb and can be used by loading it into the tensor flow service. We built the model server using the latter Tensorflow Serving. Our system only trains normal data. When abnormal data comes in as input data, it uses RMSE and recognizes it as abnormal data when RMSE is high.

3.6 Monitoring Alarm

This component alerts the user when an anomaly data detected. Figure 6 is our alert alarm web application process. Autoencoder model was trained by using PLC data from DAQ component and sensor data stored in HBase. It is a process that judges the abnormality of the data collected in real time using the Predictor component and confirms the abnormality in the web application. This component is a web application. The frontend uses the angular framework and the backend framework uses the spring boot.

3.7 RMSE-Based Warning Level

The following is a structure that notifies the user when the RMSE becomes high and recognized as an abnormal symptom. RMSE is often used as a measure to evaluate the model by obtaining the difference between the predicted value and actual value of the model with the mean square root error.

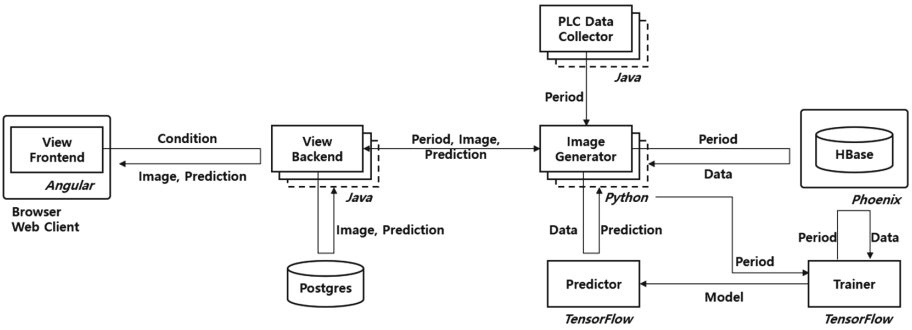

Fig. 6. Monitoring alarm process.

The formula of RMSE is as follows.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3}$$

The RMSE generated using the Autoencoder model is the error value obtained by subtracting the actual sensor data from the predicted value of the actual sensor data in the learning model. We applied the 6 sigma technique after finding the normal distribution of RMSE. Figure 7 structure is a structure that gives warning alarm to the user by using RMSE. First, if the RMSE value is higher than the level1 value, it goes to State A. If it continues to exceed the warning level line, it goes to State B and alarms the user. If there is no value beyond the warning level in State A, it goes to State C and the alarm will not sound to the user. In case of exceeding warning level value again in State C, it goes to State D and goes to State B again and gives alarm to the user.

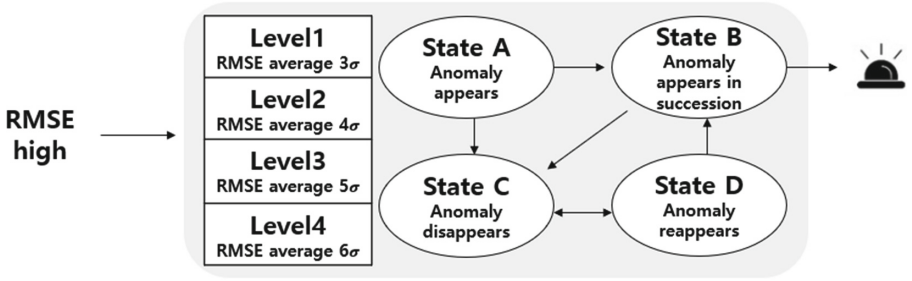

Fig. 7. Alarm process.

4 Performance Analysis

4.1 Experiment Environment

In this paper, we used TensorFlow Python library to train the model. The operating system is Ubuntu 16.04. Tensor flow supports GPU acceleration. To accelerate learning, I used four Nvidia GeForce GTX 1080 Ti graphics cards with 11 GB of RAM.

4.2 DataSet

The data set used in this paper was obtained by attaching a vibration sensor to the bearing, motor, and body of the equipment in the gold plating process of D Company. There are five facilities, and there are three types of equipment movement. Horizontal, Vertical, and Pitch. The sensor is measured in bearings, motors, and body for each movement of each facility. The learning unit learns the cycle unit of the equipment from the time when the equipment starts moving to the time when the equipment stops. The vibration sensor data generated in this process is analyzed by T-F spectrum which is a time-frequency domain by converting fast Fourier transform.

4.3 Model Result

In this paper, we compare the model performance using Restricted Boltzmann machine and Autoencoder algorithm as shown in Table 1. The model train the unit of the cycle from equipment the start to end as one input value. The input cycle value of the next time distinguishes between normal and abnormal through the difference from the reconstruction value predicted by model. The same parameters used for each model were learning rate = 0.01, Batch = 128, Epoch = 1000, and Hidden Dimension = 512.

Table 1. Comparison of the model performance.

Model	RMSE (normal)	RMSE (abnormal)
RBM	0.0192	0.0321
Autoencoder	0.0084	0.0395

Figure 8 is the result of our Autoencoder model. The first figure is the actual value of the model, and the second figure is the result of the model. The last figure is the RMSE result minus the actual value and the predicted result.

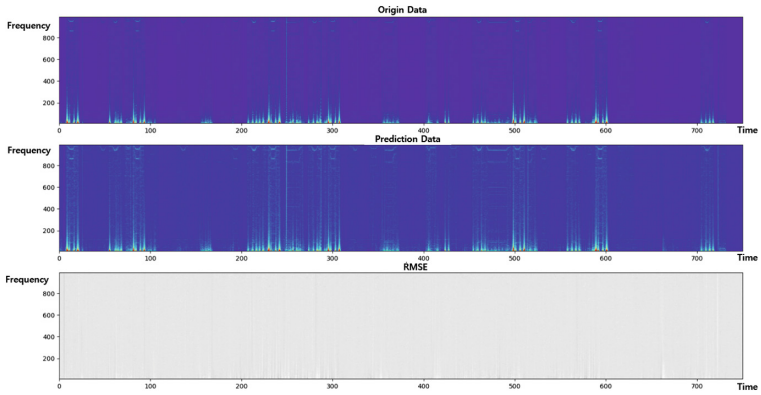

Fig. 8. Autoencoder model result.

4.4 Monitoring Result

This section is the result of using our model to alert the user. Figure 9 shows the user's current facility status using the bar chart. It is bar chart screen before actual equipment breakdown. One bar represents the 6 min RMSE value, which is the cycle time of the facility. It is a system that warns the user when the RMSE value exceeds the warning level that we set. The alarm continues to be given to the user since the level value is continuously exceeded every cycle.

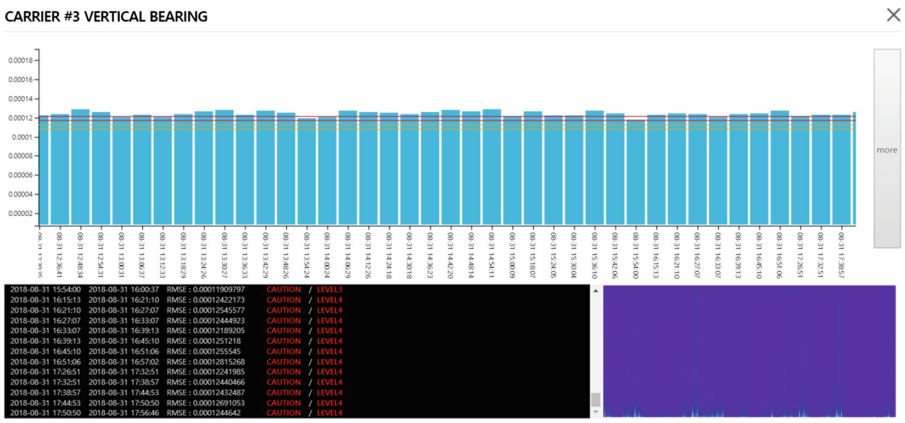

Fig. 9. Monitoring using bar chart.

5 Conclusion

In this paper, we propose a new architecture based on artificial intelligence. A platform based on a docker can be easily extended because it has good scalability.

While there are many services available on the platform, we have shown results for services related to anomaly detection. Autoencoder was used as anomaly detection model indications and anomaly indications could be detected effectively. In the next study, if the anomaly data is collected to a certain extent, it will be extended to the fault detection classification that can be found from the anomaly detection to the cause.

Acknowledgement. This work has supported by the Gyeonggi Techno Park grant funded by the Gyeonggi-Do government (No. Y181802). This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2018-0-01417) supervised by the IITP (Institute for Information communications Technology Promotion).

References

1. Ismail, B.I., et al.: Evaluation of docker as edge computing platform. In: 2015 IEEE Conference on Open Systems (ICOS), pp. 130–135 (2015)
2. Shah, J., Dubaria, D.: Building modern clouds: using docker, kubernetes & Google cloud platform. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), pp. 184–189 (2019)
3. Javaid, A., et al.: A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), pp. 21–26. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2016)
4. Du, M., et al.: Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1285–1298 (2017)
5. Kwon, D., et al.: A survey of deep learning-based network anomaly detection. Cluster Comput. 1–13 (2017)
6. Tang, T.A., et al.: Deep learning approach for network intrusion detection in software defined networking. In: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), pp. 258–263 (2016)
7. Li, W., Mahadevan, V., Vasconcelos, N.: Anomaly detection and localization in crowded scenes. IEEE Trans. Pattern Anal. Mach. Intell. **36**, 18–32 (2014)
8. Ngiam, J., et al.: Multimodal deep learning. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 689–696 (2011)
9. Andrew, G., et al.: Deep canonical correlation analysis. In: International Conference on Machine Learning, pp. 1247–1255 (2013)
10. Mohamed, A., Sainath, T.N., Dahl, G.E., Ramabhadran, B., Hinton, G.E., Picheny, M.A.: Deep belief Networks using discriminative features for phone recognition. In: ICASSP, pp. 5060–5063 (2011)
11. Erhan, D., et al.: Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. **11**, 625–660 (2010)
12. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
13. He, K., et al.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)