Self-supervised pre-training of CNNs for flatness defect classification in the steelworks industry

Filippo Galli [a,1,*], Antonio Ritacco [a,2], Giacomo Lanciano [a,3], Marco Vannocci [a,4], Valentina Colla [a,5], Marco Vannucci [a,6]

[a] Scuola Superiore Sant'Anna, Via G. Moruzzi 1, Pisa, Italy
[1] filippo.galli@santannapisa.it; [2] antonio.ritacco@santannapisa.it; [3] giacomo.lanciano@santannapisa.it;
[4] marco.vannocci@santannapisa.it; [5] valentina.colla@santannapisa.it; [6] marco.vannucci@santannapisa.it
* corresponding author

ARTICLE INFO

Article history
Received July 18, 2019
Revised August 5, 2019
Accepted October 29, 2019
Available online March 31, 2020

Keywords
Self-supervision
Steelworks
Deep learning
CNN

ABSTRACT

Classification of surface defects in the steelworks industry plays a significant role in guaranteeing the quality of the products. From an industrial point of view, a serious concern is represented by the hot-rolled products shape defects and particularly those concerning the strip flatness. Flatness defects are typically divided into four sub-classes depending on which part of the strip is affected and the corresponding shape. In the context of this research, the primary objective is evaluating the improvements of exploiting the self-supervised learning paradigm for defects classification, taking advantage of unlabelled, real, steel strip flatness maps. Different pre-training methods are compared, as well as architectures, taking advantage of well-established neural subnetworks, such as Residual and Inception modules. A systematic approach in evaluating the different performances guarantees a formal verification of the self-supervised pre-training paradigms evaluated hereafter. In particular, pre-training neural networks with the EgoMotion meta-algorithm shows classification improvements over the AutoEncoder technique, which in turn is better performing than a Glorot weight initialization.

1. Introduction

In the steelmaking cycle, continuous casting is the process where molten steel is solidified in different semi-finished products, and it is the starting point of the Hot Rolling Mill (HRM) process. Slabs are one of these intermediate products, characterized by a rectangular cross-section, and transformed into flat steel products. The primary thickness reduction of a slab can be gained via the roughing mill process where the heated slab enters, after a descaling phase, while the finishing mill process refines the thickness of the strip providing the final thickness and definitively changing the slab into a long and thin product called a strip. From an industrial point of view, a serious concern is represented by the hot-rolled products shape defects and particularly those concerning the strip flatness. Such types of defects, in fact, highlight non-uniformities within the hot rolling process but can be detected only at the end of the process and thus cannot be recovered in time before the next slab is being processed. The main consequence is the evident degradation of the quality of the final product that leads to economic losses due to non-compliant quality of the products.

Flatness defects are, on the one hand, due to different elongation in the internal strip fibre caused by uneven stress along the width or by the high rolling speed process that leads to fluttering strips. On the other hand, flatness defects are an uneven thermal gradient across the strip that is responsible for flatness

defects generating waviness. Uneven heating or cooling process is the main cause of the latter type of defects due to internal stresses that can locally overcome the yield stress of the material leading to plastic deformation of the strip [1] [2]. Defects due to different elongation of the fibre are particularly relevant, as they are directly connected to rolling process parameters such as the inflection of the working rolls (bending) or the relative sliding of the work rolls along the transverse axis (shifting).

Flatness defects are typically divided into two sub-classes, depending on whether the edge of the strip is affected or not. When the edge is affected, the defect is typically referred to as a "wave defect," while a buckle typically refers to a defect that does not affect the strip edge. In addition, the position along the transverse direction of the strip allows categorizing buckles in center- or quarter-buckles. In the former case, the defect occurs near the longitudinal centerline of the strip, while in the latter case, it occurs in the transverse regions that engage the upper/lower strip at a distance of about one-quarter of the width from the strip edge.

The strip planarity is usually measured by considering the strip as formed by a series of adjacent longitudinal fibres: if all the fibres have the same length, the strip is perfectly flat. The presence of flatness defects derives from the fact that the fibres do not stretch independently, and when they have different lengths, flatness defects appear as waves on the strip. The main parameter used for the numerical evaluation of strip flatness is the so-called I-Unit index, which is computed for each fibre as follows:

$$I - Unit\ (i) = \left(\frac{L_i - L_{ref}}{L_{ref}}\right) \cdot 10^5 \quad (1)$$

where L_i is the length of fibre i, and L_{ref} is the length of a reference fibre. Typically, the reference fibre is the shortest one, and the I-Unit assumes only non-negative values.

The transversal flatness profiles of each strip are usually concatenated and represented as a bi-dimensional map of the strip flatness, which is read directly from a measuring system installed at the end of the finishing mill. The procedure to detect and isolate each defect on the strip surface, which is detailed in [3], provides defect sub-images from the full strip image. HRM surface defects classification was tackled in recent years by exploiting Support Vector Machines (SVM) [4], supervised Neural Network with Back Propagation [5], unsupervised classifiers via Self-Organizing Maps (SOM) [6], or Learning Vector Quantiser (LVQ)[7].

In general, industrial surface defects detection and classification systems currently applied in the steel sector exploit Artificial Intelligence-based approaches at different levels: in the preliminary pre-processing stage, for instance for removal of unreliable data [8] and feature selection [9][10] as well as in the actual classification stage [11]–[13]. Moreover, machine learning approaches are applied to correlate the different kinds of defects with their potential causes [14][15]. However, in this context, the potential of high capacity networks has not yet been fully exploited. Very recently, in [3], the use of Convolutional Neural Network (CNN) is also introduced to cope with the classification problem of surface defects.

In this paper, the classification problem is extended and tackled from a different point of view. In particular, we explore the idea of using unsupervised pre-training of CNNs, which does not require manual labeling of a dataset. Later fine-tuning on a labelled dataset via transfer learning lets us compare the effectiveness of the considered methods to increase classification accuracy over the use of mere supervised learning

2. Method

2.1. Self-Supervised Learning Techniques

High capacity networks are solving many different machine learning tasks, ranging from large-scale image classification [16], segmentation [17], and image generation [18] to natural speech understanding

[19] and realistic text-to-speech [20]. A few general trends are easily identified in academia and industry: deeper networks show increasingly better results [21] as long as they are fed with ever-larger amounts of data, and labelled data in particular. Computational and economic costs increase linearly with the size of the dataset. For this reason, in the latest years, some unsupervised approaches were aimed at the exploitation of unlabelled data. The intuition behind many of these techniques was emulating the human brain's ability to self-determine the task goal and to improve it.

Advancements in algorithms able to exploit labels inherently contained within an unlabelled dataset gave rise to what is now referenced as self-supervised learning. LeNet-5 [22] popularized convolutional operators by embedding apriori knowledge of the data into networks by preserving the spatial correlation of the pixel of an image as the signal proceeds through the layers of the network itself. Similarly, self-supervision embeds apriori knowledge about a dataset into a network, but not by introducing a different operator. Instead, the output of the network is typically constrained to be coherent with a known transformation of the inputs. Since the input and the transformations are known, we can picture this situation as deriving labels from the input data and forcing the network to converge to those labels. Assuming weights learned through self-supervised learning generalize to a similar task, one can use transfer learning [23][24] to fine-tune the network on a labelled dataset. A few examples of self-supervised techniques include:

a) Physics and Domain Knowledge [25]: The authors show how a CNN fed with images of a video stream of a falling ball learns to predict the height of a falling object, just by forcing the output to be coherent with the coordinates of a parabola, which is the physically feasible trajectory of a falling body.

b) Unsupervised Jigsaw Puzzles [26]: quoting the authors "By following the principle of self-supervision, the authors build a CNN that can be trained to solve jigsaw puzzles as a pretext task, which requires no manual labeling. The CNN is later repurposed to solve classification and detection via transfer learning".

c) Colorization [27]: the auxiliary task is to predict two color channels of an image. It has given the luminosity of each pixel. Also, the representations of the internal feature are learned by colorizing unlabelled images, which can be fine-tuned for classification and detection.

The above methods could not be exploited because: a) our system does not provide a video stream; b) classified objects do not have strong structural properties that identify each shape and c) images are greyscale, not having color channels other than luminosity. Conversely, the method proposed by Agrawal *et al.* [28] investigates if the awareness of EgoMotion could be used as a supervisory signal for feature learning. In other words, images of a moving item show different instances of the same object, i.e., a fixed label for different samples. Edges, texture, and colors needed to recognize the object are visual features that persist independently of the location of the object itself.

One way to emulate the situation of learning via EgoMotion is to:

1) Present a (bottom) CNN with the image of an unlabelled object and let it output a $w \cdot h \cdot f$ tensor where f is the number of filters of the last layer, and w, h are the width and height of the feature maps.

2) Feed the same network a randomly transformed version of the same image, by translating/rotating it and let it output a new tensor $w \cdot h \cdot f$.

3) Concatenate the two outputs to form a $w \cdot h \cdot 2f$ tensor and feed it to a (top) CNN tasked with predicting the random transformation, which is known, and constitutes the label.

The schematics of the network is shown in Fig. 1.

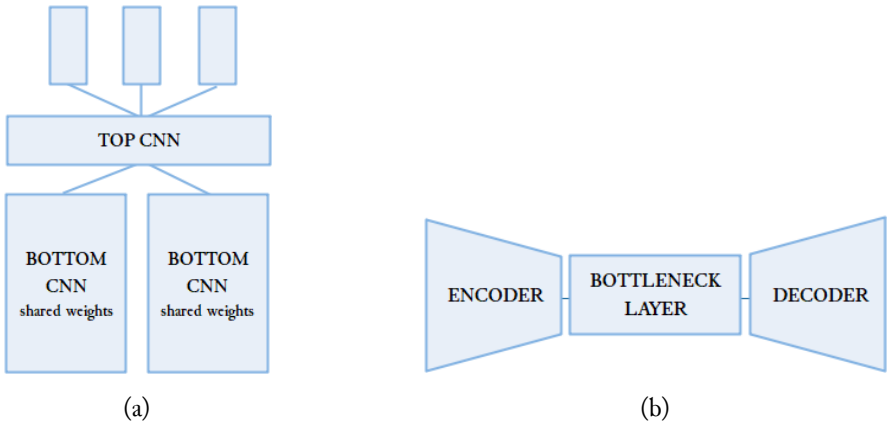

Fig. 1. The EgoMotion (a) and AutoEncoder learning architectures (b)

The self-supervised learning problem is framed as a supervised learning problem and, by backpropagating and iterating over the unlabelled dataset, the solution is a network that exploits visual features to predict the random transformations applied to the image. It is a reasonable assumption that these features can instead be repurposed to classify an image, via transfer learning, which is the goal of our approach.

Another example of a self-supervised technique for learning considered in this work is that of autoencoders, which consist of a neural network that tries to learn the identity function $h_{W,b}(x) \approx x$ [29]. Without placing some form of information bottleneck inside the function h, the task of learning the identity function would be trivial. Instead, the amount of information that passes through the network is reduced by having layers with smaller representation capacity, in a way that allows projecting input data in a latent space characteristic of the training data. As the autoencoder is forced to prioritize which aspects of the input should be transferred, it often learns useful properties of the data. Autoencoders are typically composed of two parts: An encoder, that takes the input and generates the latent encoding, and a decoder, that takes the latent encoding and generates the reconstruction of the input. Depending on the task at hand, a different type of autoencoders can be used, for instance:

a) Under complete autoencoders [30]: the latent space representation in the bottleneck layer is achieved, constraining the dimension of the output of the encoder to be smaller than the dimension of the input by placing less hidden units than input units.

b) Regularized autoencoders [31]: a loss function with regularization used to encourage the model to have representation sparsity (Sparse Autoencoders [32]) and robustness noise/missing inputs (Denoising Autoencoders [33]), rather than limiting the model to reduce the hidden units number.

Since we are dealing with images and we need to reduce the image representation to a tensor coherent with the one produced by the networks pre-trained with EgoMotion, under-complete Convolutional AutoEncoders represent a reasonable solution. These models present a series of convolutional and max-pooling layers to reduce the input to a certain encoding. While resorting is used to transpose convolutional and up-sampling layers for decoding.

2.2. Architectures

Throughout the experiments, we used a repeating pattern to develop network architectures of different representational capacity. Independently of the self-supervision method applied for pre-training, every network shares the same type of layers. Specifically, we built two modules:

1) Inception module: based on Szegedy *et al.* [34], we derived an inception layer where the input branches out to four convolutional modules with different kernel sizes, such as the one reported in Fig. 2(a).

2) Residual module: similarly, based on He *et al.* [35], we defined a residual layer where the input undergoes heavier convolutional processing on one path while being left almost untouched on another path. Both signals are summed to produce the module's output, as shown in Fig. 2(b).

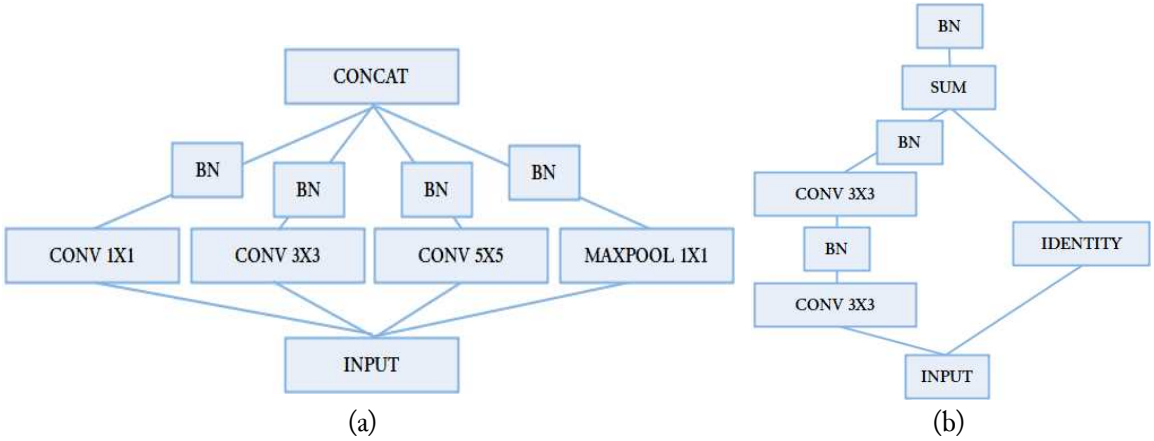

Fig. 2.– Inception (a) and Residual (b) modules

Both modules provide the possibility to apply batch normalization, as well as different convolutional strides. We define a number of base networks composed of the above modules, and the corresponding naming convention is, e.g., EMInc4BN for a network of four Inception modules with Batch Normalization and pre-trained with EgoMotion, and AERes8 for a network of 8 Residual modules without Batch normalization and pre-trained with AutoEncoders. Table 1 shows the structure of 4 models with increasing complexity. Each structure is composed of either Inception or Residual modules, for a total of 8 networks. They have been trained with and without batch normalization, totaling to 16 models.

Table 1. The network structure for different depths: C64s4 means Inception or Residual module with 64 filters and 4x4 strides. MP4 means the Max Pooling layer with 4x4 windows and strides. The output of each model is a 7x7x64 tensor of 64 7x7 feature maps

Depth	Base Structure
8	C64→C64→MP2→C128→C128→MP2→C128→C128→MP2→C64→C64→MP2
4	C64→MP2→C128→MP2→C128→MP2→C64→MP2
2	C128s2→MP2→C64s2
1	C64s4→MP4

3. Results and Discussion

3.1. Experiments

Every single model undergoes three different training techniques, such as pre-training with EgoMotion and transfers learning on the classification dataset, pre-training with AutoEncoder, and transfer learning on the classification dataset, and training from scratch on the classification dataset.

During transfer learning, every layer is left trainable, and weights learned during pre-training were not frozen when turning to classification. In the context of pre-training with EgoMotion, each one of the base networks described in Section 2.1 constitutes the bottom CNN, while the top CNN consists of a dense layer of 300 ELU units [36], followed by a 0.3 rate dropout and the output layer. This learning technique requires three outputs: one for predicting rotations and two for vertical and horizontal translations. As in Agrawal *et al.* [28], the problem is framed as a classification task, so every output is an array of softmax units predicting the bin corresponding to the right transformation.

In the context of pre-training with AutoEncoders, each one of the base networks described in Section 2.2 constitutes the encoding part, which outputs a 7x7x64 tensor. The decoder architecture is common to each model and is composed of 5 transposed convolutional layers preceded by up-sampling layers. The

last convolution has sigmoid activation functions, which are a good fit for regressing pixel luminosity values scaled to the 0-1 range.

Every network is trained using Adam [37] optimizer for 100 epochs with early stopping and L2 regularization to prevent overfitting. Once pre-training is completed, every network is repurposed for classification by removing either the top CNN or the decoder for EgoMotion and AutoEncoders, respectively, and by plugging a 0.3 rate dropout layer, a 20 ELU unit dense layer, and a final four softmax unit layer. Adam optimizer was run for 100 epochs every 64-sample batch, and training was terminated with early stopping. Heavy artificial data augmentation was part of the process, applying random affine transformations to the input images, such as horizontal and vertical flip, width and height shift, and zooming. Similarly, the training process for classification was also carried out without pre-training of the networks and using Glorot initialization [38]. In order to have better confidence in the performance scores, training on the classification dataset was run three times, and the results averaged.

3.2. Dataset

In this work, we exploit the data used in Vannocci *et al.* [3] for what concerns the labelled dataset, where a thorough explanation of how the built dataset is presented. Using the same data, we can compare pre-training techniques against a common baseline to establish the effectiveness of self-supervision. Here we propose a summary of the main features of the exploited dataset.

Defect images are extracted from the overall image of the strip and manually classified in 4 different categories - Wave, Buckle, Multiwave, and Multibuckle (see Fig. 3) by expert personnel. Every strip image is affected by a varying number of defects, so dataset splits refer to defect images, not the strip images. Of these, ~80% is devoted to the training and validation sets, while the remaining ~20% are test images. This results in a dataset composed of 4806 images: 3938 images were used for training and validation in a 70-30% split, 868 images were used for testing. The class distribution is shown in Table 2. For what concerns the data used for self-supervised training, we ran the bounding box algorithm in Vannocci *et al.* [3] on new strips to recover 32437 new unlabelled defect images.

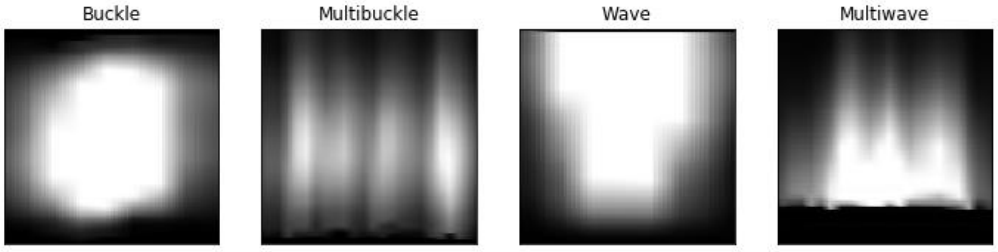

Fig. 3. Flatness defect samples

Table 2. Distribution of flatness defect classes in the labeled dataset [3]

Class	TR&VD	TS
Wave	1209	225
Buckle	1201	261
Multiwave	840	179
Multibuckle	688	203
Total	***3938***	***868***

3.3. Results

The results of all experiments are summarized in Table 3 and Table 4. Fig. 4 showed each model's performance, comparing the accuracy of the same model with different pre-training policies. In the vast majority of models, we see an increase in validation accuracy whenever pre-training occurs. Pre-training with EgoMotion almost always guarantees a better classification accuracy over training from scratch,

where initial weights are initialized with Glorot [38]. Specifically, the overall average accuracy increase is equal to 1.03%, which for a validation accuracy of 90%, would mean a relative decrease in the error rate of about 10%. Similarly, pre-training with AutoEncoders shows a performance increase when the model is simpler - typically when the model is 1 or 2 modules deep. The overall average accuracy increase is still positive and equal to 0.41%.

Table 3. Validation accuracies on the classification dataset for NN with **Inception** modules. With and without Batch Normalization, and with weight initialization via EgoMotion, AutoEncoder, or Glorot. The total parameter number is also included.

		BN-Depth				No BN-depth			
		1	*2*	*4*	*8*	*1*	*2*	*4*	*8*
EgoMotion	*1*	0.8864	0.8892	0.9195	0.9141	0.8631	0.8909	0.9195	0.9123
	2	0.8614	0.8837	0.9052	0.9204	0.8515	0.8962	0.9168	0.9097
	3	0.8685	0.8703	0.9141	0.9213	0.8426	0.8909	0.924	0.9079
Glorot	*1*	0.88524	0.8614	0.9079	0.9114	0.7916	0.8837	0.9079	0.9044
	2	0.8301	0.8007	0.9258	0.9195	0.8229	0.8819	0.915	0.9186
	3	0.839	0.856	0.9177	0.9168	0.7934	0.8694	0.9177	0.9123
AutoEncoder	*1*	0.8519	0.8976	0.912	0.9162	0.8528	0.8756	0.909	0.9078
	2	0.8596	0.8935	0.9069	0.9146	0.846	0.8858	0.9019	0.9086
	3	0.8689	0.8976	0.9061	0.9154	0.8376	0.8909	0.9044	0.9078
#Parameters		63,720	138,660	360,00	730,782	63,404	137,000	358,000	727,000

Table 4. Validation accuracies on the classification dataset for NN with Residual modules. With and without Batch Normalization, and with weight initialization via EgoMotion, AutoEncoder, or Glorot. The total parameter number is also included.

		BN-Depth				No BN-depth			
		1	*2*	*4*	*8*	*1*	*2*	*4*	*8*
EgoMotion	*1*	0.8989	0.9249	0.9106	0.907	0.8891	0.9061	0.9195	0.9141
	2	0.881	0.9114	0.9222	0.9177	0.8775	0.9302	0.9106	0.9186
	3	0.8989	0.9106	0.9258	0.9213	0.9034	0.9159	0.9132	0.9132
Glorot	*1*	0.8971	0.9114	0.9034	0.9007	0.873	0.9061	0.9079	0.9168
	2	0.8846	0.9097	0.9141	0.9088	0.8739	0.9052	0.9123	0.915
	3	0.8846	0.9088	0.9123	0.9052	0.8739	0.9159	0.9195	0.9114
AutoEncoder	*1*	0.8968	0.912	0.9154	0.9129	0.8646	0.9078	0.9108	0.9069
	2	0.8934	0.9044	0.9112	0.9078	0.8942	0.9052	0.9069	0.8968
	3	0.89	0.8968	0.9188	0.9078	0.8799	0.9095	0.9019	0.9069
#Parameters		101,544	333,992	350,568	1,494,760	100,520	330,920	744,484	1,482,472

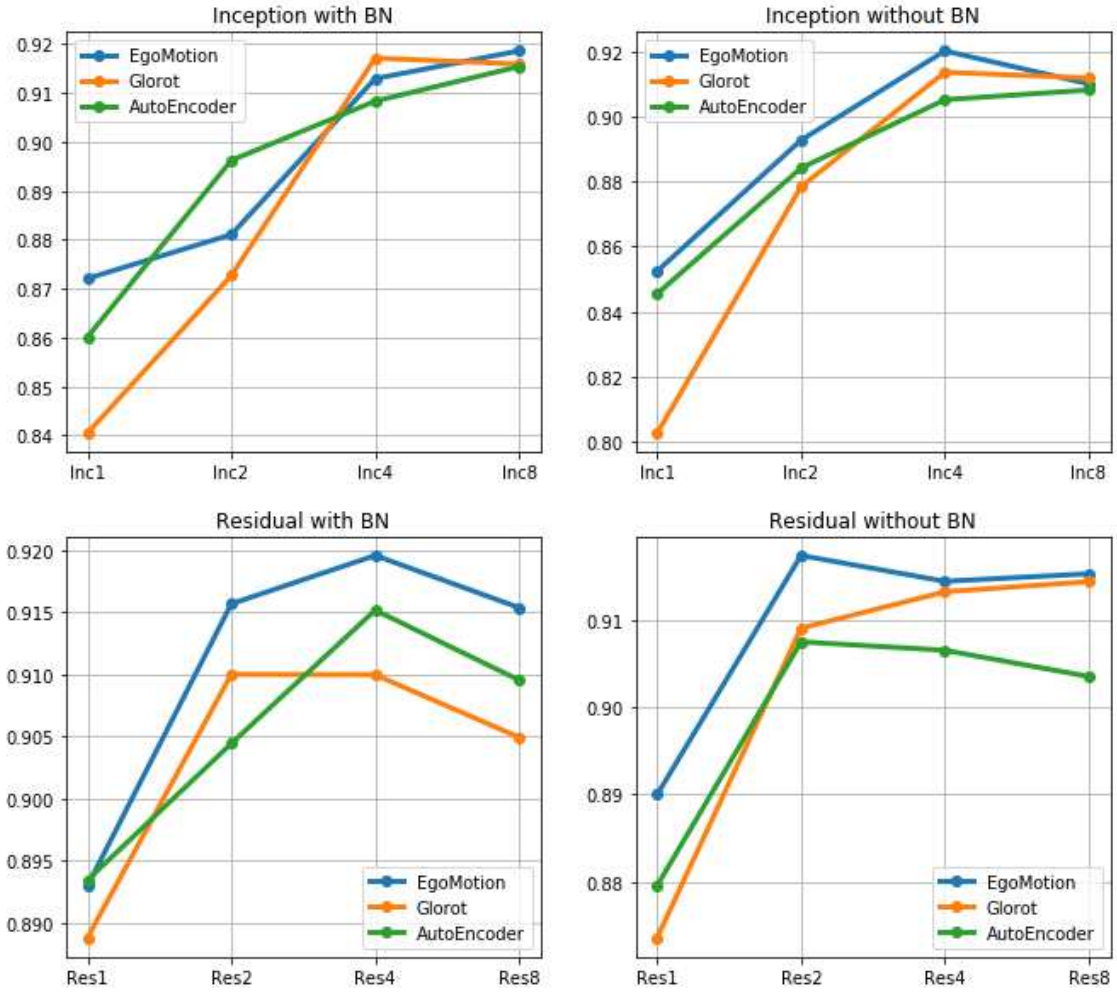

Fig. 4. Average validation accuracies over three runs for each model and pre-training policy.

4. Conclusion

The problem of classification of surface defects in the steel industry has been examined and advanced in this study by exploiting unlabelled data. By doing so, the improvements in classification accuracy come without a corresponding increase in costs due to expert personnel devoted to assembling a bigger labelled dataset. In particular, we have shown that using self-supervised learning algorithms for pre-training different Convolutional Neural Network architectures leads to increased accuracy once the models are fine-tuned via transfer learning on the classification task. Concerning similar results on Vannocci *et al.* [3] on the same classification dataset, we underline four major achievements. The first, validation accuracy is generally improved, with the best performing network EMRes2 outperforming the results of previous research, 92.7% to 93.0%. Second, All the models evaluated in this context have drastically reduced the number of parameters needed to achieve a comparable - if not better - performance. EMRes2 has more than 160-times fewer parameters than Inception 311 in Vannocci *et al.* [3]. Third, the accuracy of EMRes2 (90.6%) showed overfitting signs, but it still increased with respect to Inception311 (89.2%). At last, we can conclude the increase in accuracy comes without the need of additional labelled images, by adopting self-supervised algorithms for pre-training.

References

[1] V. B. Ginzburg, *Flat Rolling Fundamentals*, 2000, doi: 10.1201/9781482277357.

[2] A. Bhaduri, "Rolling," 2018, doi: 10.1007/978-981-10-7209-3_12.

[3] M. Vannocci *et al.*, "Flatness Defect Detection and Classification in Hot Rolled Steel Strips Using Convolutional Neural Networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2019, doi: 10.1007/978-3-030-20518-8_19.

[4] G. Wu, H. Kwak, S. Jang, K. Xu, and J. Xu, "Design of online surface inspection system of hot rolled strips," in *Proceedings of the IEEE International Conference on Automation and Logistics, ICAL 2008*, 2008, doi: 10.1109/ICAL.2008.4636548.

[5] S. Ghorai, A. Mukherjee, M. Gangadaran, and P. K. Dutta, "Automatic defect detection on hot-rolled flat steel products," *IEEE Trans. Instrum. Meas.*, 2013, doi: 10.1109/TIM.2012.2218677.

[6] P. Caleb and M. Steuer, "Classification of surface defects on hot rolled steel using adaptive learning methods," *Int. Conf. Knowledge-Based Intell. Electron. Syst. Proceedings, KES*, 2000, doi: 10.1109/kes.2000.885769.

[7] G. Wu, K. Xu, and J. Xu, "Application of a new feature extraction and optimization method to surface defect recognition of cold rolled strips," *J. Univ. Sci. Technol. Beijing Miner. Metall. Mater. (Eng Ed)*, 2007, doi: 10.1016/S1005-8850(07)60086-3.

[8] S. Cateni, V. Colla, and G. Nastasi, "A multivariate fuzzy system applied for outliers detection," *J. Intell. Fuzzy Syst.*, 2013, doi: 10.3233/IFS-2012-0607.

[9] S. Cateni, V. Colla, and M. Vannucci, "A hybrid feature selection method for classification purposes," in *Proceedings - UKSim-AMSS 8th European Modelling Symposium on Computer Modelling and Simulation, EMS 2014*, 2014, doi: 10.1109/EMS.2014.44.

[10] S. Cateni, V. Colla, and M. Vannucci, "A genetic algorithm-based approach for selecting input variables and setting relevant network parameters of a SOM-based classifier," *Int. J. Simul. Syst. Sci. Technol.*, 2011, available at: Google Scholar.

[11] A. Borselli, V. Colla, M. Vannucci, and M. Veroli, "A fuzzy inference system applied to defect detection in flat steel production," in *2010 IEEE World Congress on Computational Intelligence, WCCI 2010*, 2010, doi: 10.1109/FUZZY.2010.5584036.

[12] A. Borselli, V. Colla, and M. Vannucci, "Surface defects classification in steel products: A comparison between different artificial intelligence-based approaches," in *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Applications, AIA 2011*, 2011, doi: 10.2316/P.2011.717-068.

[13] M. Vannucci, V. Colla, M. Sgarbi, and O. Toscanelli, "Thresholded Neural Networks for Sensitive Industrial Classification Tasks," 2009, pp. 1320–1327, doi: 10.1007/978-3-642-02478-8_165.

[14] J. Brandenburger, V. Colla, G. Nastasi, F. Ferro, C. Schirm, and J. Melcher, "Big Data Solution for Quality Monitoring and Improvement on Flat Steel Production," *IFAC-PapersOnLine*, 2016, doi: 10.1016/j.ifacol.2016.10.096.

[15] M. Appio, A. Ardesi, and A. Lugnan, "Automatic surface inspection in steel products ensures safe, cost-efficient and timely defect detection in production," in *AISTech - Iron and Steel Technology Conference Proceedings*, 2018, doi: 10.5151/1983-4764-31378.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015, available at: Google Scholar.

[17] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018, doi: 10.1109/TPAMI.2017.2699184.

[18] A. Creswell and A. A. Bharath, "Denoising Adversarial Autoencoders," *IEEE Trans. Neural Networks Learn. Syst.*, 2019, doi: 10.1109/TNNLS.2018.2852738.

[19] A. Kumar *et al.*, "Ask me anything: Dynamic memory networks for natural language processing," in *33rd International Conference on Machine Learning, ICML 2016*, 2016, available at: Google Scholar.

[20] S. Arik *et al.*, "Deep voice: Real-time neural text-to-speech," in *34th International Conference on Machine Learning, ICML 2017*, 2017, available at: Google Scholar.

[21] H. Mhaskar, Q. Liao, and T. Poggio, "When and why are deep networks better than shallow ones?," in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 2017, available at: Google Scholar.

[22] S. Wu, W. Wei, and L. Zhang, "Comparison of machine learning algorithms for handwritten digit recognition," in *Communications in Computer and Information Science*, 2018, doi: 10.1007/978-981-13-1651-7_47.

[23] S. J. Pan and Q. Yang, "A survey on transfer learning," 2010, doi: 10.1109/TKDE.2009.191.

[24] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, doi: 10.1007/978-3-030-01424-7_27.

[25] R. Stewart and S. Ermon, "Label-free supervision of neural networks with physics and domain knowledge," in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 2017, available at: Google Scholar.

[26] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, doi: 10.1007/978-3-319-46466-4_5.

[27] G. Larsson, M. Maire, and G. Shakhnarovich, "Colorization as a proxy task for visual understanding," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, doi: 10.1109/CVPR.2017.96.

[28] P. Agrawal, J. Carreira, and J. Malik, "Learning to see by moving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, doi: 10.1109/ICCV.2015.13.

[29] D. H. Ballard, "Modular Learning in Neural Networks," *Aaai*, 1987, available at: Google Scholar.

[30] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, 2009, doi: 10.1561/2200000006.

[31] M. Guarascio, G. Manco, and E. Ritacco, "Deep learning," 2018, doi: 10.1016/B978-0-12-809633-8.20352-X.

[32] A. Makhzani and B. Frey, "k-Sparse autoencoders," in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014, available at: Google Scholar.

[33] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," *J. Mach. Learn. Res.*, 2010, available at: Google Scholar.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, doi: 10.1109/CVPR.2016.308.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, doi: 10.1109/CVPR.2016.90.

[36] D. A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016, available at: Google Scholar.

[37] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015, available at: Google Scholar.

[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Journal of Machine Learning Research*, 2010, available at: Google Scholar.