machine learning & knowledge extraction

MDPI

Article

A Transfer Learning Evaluation of Deep Neural Networks for Image Classification

Nermeen Abou Baker *, Nico Zengeler and Uwe Handmann

Computer Science Institute, Ruhr West University of Applied Sciences, 46236 Bottrop, Germany; Nico.Zengeler@hs-ruhrwest.de (N.Z.); uwe.handmann@hs-ruhrwest.de (U.H.)
* Correspondence: nermeen.baker@hs-ruhrwest.de

Abstract: Transfer learning is a machine learning technique that uses previously acquired knowledge from a source domain to enhance learning in a target domain by reusing learned weights. This technique is ubiquitous because of its great advantages in achieving high performance while saving training time, memory, and effort in network design. In this paper, we investigate how to select the best pre-trained model that meets the target domain requirements for image classification tasks. In our study, we refined the output layers and general network parameters to apply the knowledge of eleven image processing models, pre-trained on ImageNet, to five different target domain datasets. We measured the accuracy, accuracy density, training time, and model size to evaluate the pre-trained models both in training sessions in one episode and with ten episodes.

Keywords: transfer learning; image classification; deep neural network

Citation: Abou Baker, N.; Zengeler, N.; Handmann, U. A Transfer Learning Evaluation of Deep Neural Networks for Image Classification. *Mach. Learn. Knowl. Extr.* **2022**, *4*, 22–41. https://doi.org/10.3390/make4010002

Academic Editor: Andreas Holzinger

Received: 3 December 2021
Accepted: 10 January 2022
Published: 14 January 2022

1. Introduction

Deep learning is a subfield of machine learning that allows computers to automatically interpret representations of data by learning from examples. Transfer learning is a deep learning technique that uses previous knowledge to learn new tasks and is becoming increasingly popular in many applications with the support of Graphics Processing Unit (GPU) acceleration. Transfer learning has many benefits that have attracted researchers in different domains, to name but a few: medical applications [1], remote sensing [2], optical satellite images [3], supporting automated recycling [4], natural language processing [5], mobile applications [6], etc. However, there are some caveats in choosing the best pre-trained model for such applications, as most focus on accuracy and leave out other important parameters. Therefore, it is important to also consider other metrics such as training time or memory requirements before proceeding to a concrete implementation.

Transfer learning is performed with pre-trained models, typically large Convolutional Neural Networks (CNNs) that are pre-trained on large standard benchmark datasets and then reused for the new target task. The reuse of such pre-trained models can be easily implemented by, for example, replacing certain layers with other task-specific layers and then training the model for the target task. Moreover, many frameworks such as PyTorch, MATLAB, Caffe, TensorFlow, Onnx, etc., provide several pre-trained models that can help researchers implement this promising technique. The state-of-the-art has many architectures, each with its own characteristics, that are suitable for CNN applications. However, the performance of the resulting transfer learning network depends on the pre-trained model used. Before going into the reuse of these models, it seems that there is a great deal of freedom in choosing the model.

According to [7], the size and similarity of the target dataset and the source task can be used as rules of thumb to choose the pre-trained model. ImageNet is a leading dataset due to its popularity and data diversity. However, fine-tuning pre-trained models that are trained on ImageNet is not per se able to achieve good results on spectrograms, for example.

Besides, following the previous strategy might not be enough with the current challenging constraints that require high accuracy, a short training time, and limited hardware resources for specific applications. Previously pre-trained model analysis was presented in [8], who collected reported values from the literature and compared the models' performance on ImageNet to evaluate several scores, such as the top-five accuracy normalized to model complexity and power consumption.

Another worthwhile attempt was presented by [9], who benchmarked pre-trained models on ImageNet using multiple indices such as accuracy, computational complexity, memory usage, and inference time to help practitioners better fit the resource constraints.

Choosing the best pre-trained model is a complex dilemma that needs to be well understood, and researchers could feel confused about picking the most suitable option. We performed extensive experiments to classify five datasets on eleven pre-trained models. We provide in-depth insight and offer a feasible guideline for transfer learning that uses a pre-trained model by introducing an overview of the tested models and datasets and evaluating their performance using different metrics. Since most pre-trained models are used to classify ImageNet, we conducted our research on different datasets, including standard and non-standard tasks.

The paper is organized as follows: It starts by introducing the research gap in the Introduction in Section 1. Section 2 summarizes the related learning methods. Section 3 gives an overview of the main characteristics of the tested models and datasets. Section 4 focuses on the implementation of the models. Results are presented and discussed in Section 5. Finally, the conclusion of the work is given in Section 6.

2. Summary of Related Learning Methods

Machine learning is data-hungry; therefore, it has tremendous success in data-intensive applications, but it is limited when the dataset is small. This section summarizes different types of related machine learning methods for solving image classification tasks, including zero-shot learning, one-shot learning, few-shot learning, and transfer learning. One common advantage of these methods is that they leave out the burden of collecting large-scale supervised data and the issue of data scarcity.

2.1. Zero-Shot Learning

With zero-shot learning, it is possible to train a model without accessing data with non-observed labels during training by using previous labels and some auxiliary information. It assumes that the model can classify instances of unseen visual examples. This method looks promising when new unlabeled examples are introduced frequently [10]. In the zero-shot learning method, the test set and training class set are disjoint [11]. Several solutions to this problem have been proposed, such as learning intermediate attribute classifiers [12], learning a mixture of seen class proportions [13], or compatibility learning frameworks [14], for example.

2.2. One-Shot Leaning

One of the limitations of deep learning is that it demands a huge amount of training data examples to learn the weights. However, one-shot learning seeks to predict the required output based on one or a few learning examples. However, this is usually achieved by either sharing feature representations [15] or model parameters [16]. Methods such as this are useful for classification tasks when it is hard to classify data for every possible class or when new classes are added [10]. One-shot learning has been proven to be an efficient method as the number of known labels grows because in this case, it is most likely that the model has already learned a label that is very similar to the one to be learned [17].

2.3. Few-Shot Learning

This method refers to feeding a model with a small number of training data samples. It is useful for applications that lack information or can be accessed only with difficulty due to concerns about privacy, safety, or ethical issues [18].

2.4. Transfer Learning

In line with the previously mentioned methods, and according to [19–21], transfer learning methods often use few-shot learning, where prior knowledge is transformed from the source task into a few-shot task [19]. There are two ways to implement transfer learning: fine-tuning only the classifier layers, which keeps the entire model's weight constant, excluding the last layer, and fine-tuning all layers, which allows the weights to change throughout the entire network. Section 4 describes these two ways technically [22].

3. Models and Datasets

In this section, we present the CNN-design-based architectures as a critical factor in constructing the pre-trained models, the tested models, and the datasets.

3.1. CNN-Design-Based Architectures

The CNN is the fundamental component in developing a pre-trained model, and to understand the architecture, some criteria define the design architecture of the models, as follows:

- Depth: The NN depth is represented by the number of successive layers. Theoretically, deep NNs are more efficient than shallow architectures, and increasing the depth of the network by adding hidden layers has a significant effect on supervised learning, particularly for classification tasks [23]. However, cascading layers in a Deep Neural Network (DNN) is not straightforward, and this may cause an exponential increase in the computational cost;
- Width: The width of a CNN is as significant as the depth. Stacking layers may learn various feature representations, but they would not learn useful features. Therefore, a DNN should be wide enough, so the loss at the local minima could be smaller with larger layer widths [24];
- Spatial kernel size: A CNN has many parameters and hyperparameters, including weights, biases, the number of layers, the activation function, the learning rate, and the kernel size, which define the level of granularity. Choosing the kernel size affects the correlation of neighboring pixels. Smaller filters extract local and fine-grained features, whereas larger filters extract coarse-grained features [25];
- Skip connection: Although a deeper NN yields better performance, it may face challenges in performance degradation, vanishing gradients, or higher test and training errors [26]. To tackle these problems, the shortcut layer connection was first proposed by [27] by skipping some intermediate layers to allow the special flow of information across the layers, for example zero-padding, projection, dropout, skip connections, etc.;
- Channels: CNNs have powerful performance in learning features automatically, and this can be dynamically performed by tuning the kernel weights. However, some feature maps have little or no role in object discrimination [28] and could cause overfitting as well. Those feature maps (or the channels) can be optimally selected in designing the CNN to avoid overfitting.

3.2. Neural Network Architectures

This study tested eleven popular pre-trained models. Figure 1 gives a comprehensive infographic representation over time. Table 1 depicts all the tested models with their main characteristics based on their design, which is discussed in Section 3.1.

Table 1. The tested models with their main characteristics, where * refers to features specially designed for the model.

Model	Year	Depth	Main Design Characteristics	Reference
AlexNet	2012	8	Spatial	[29]
VGG-16	2014	16	Spatial and depth	[30]
GoogLeNet	2014	22	Depth and width	[31]
ResNet-18	2015	18	Skip connection	[32]
SqueezeNet	2016	18	Channels	[33]
ResNext	2016	101	Skip connection	[34]
DenseNet	2017	201	Skip connection	[35]
MobileNet	2017	54	Depthwise separable conv *	[36]
WideResNet	2017	16	Width	[37]
ShuffleNet-V2	2017	50	Channel shuffle *	[38]
MnasNet	2019	No linear sequence	Neural architecture search *	[39]

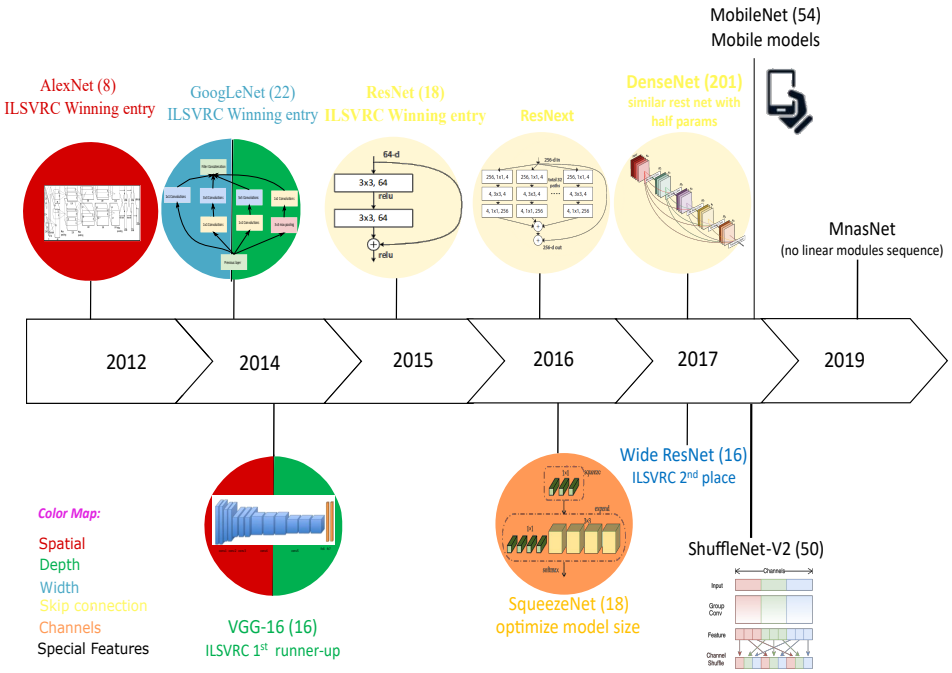

Figure 1. Infographic of the tested pre-trained models. Each model is introduced with its architecture symbol, the number of layers between brackets, and design specification (see the color map).

3.3. Datasets

A combination of standard datasets was tested, which were: CIFAR10 with 60 K images [40], Modified National Institute of Standards and Technology (MNIST) with 70 K images [41], Hymenoptera [42], and non-standard, which were: smartphones and augmented smartphones [43], as follows:

3.3.1. Hymenoptera

This is a small RGB dataset that is used to classify ants and bees from a PyTorch tutorial on transfer learning. It consists of 245 training images and 153 testing images.

3.3.2. Smartphone Dataset

This is a relatively small dataset of different smartphone models, representing six brands, namely: Acer, HTC, Huawei, Apple, LG, and Samsung. It contains 654 RGB images with twelve classes, which are: Acer Z6, HTC 12S, HTC R70, Huawei Mate 10, Huawei P20, iPhone 5, iPhone 7 Plus, iPhone 11 Pro Max, LG G2, LG Nexus 5, Samsung Galaxy S20 Ultra, and Samsung S10E. We created this dataset as a case study in a previous work [43],

to show that transfer learning can reach high accuracy with a small dataset to support automated e-waste recycling through device classification. We collected the images from the search engines focusing on the backside where unique features such as the logo and camera lenses, which are distinguishing because most front-sides of modern smartphones look similar, as showcased in Figure 2.

Figure 2. Example of a subset of the smartphone dataset.

3.3.3. Augmented Smartphone Dataset

Data augmentation is usually used to increase the volume of the dataset effortlessly. We applied a rotation operation in combination with increasing the noise. For the rotation operation, we rotated by $r \in \{45°, 135°, 225°, 315°\}$; for the noise operation, we added noise in percentages $p \in \{10\%, 25\%, 50\%\}$ by adding pixels from a discrete uniform distribution $\{0 \ldots 255 \cdot p\}$. This resulted in a total of twelve augmentation operations for each image. Therefore, the total number of images was multiplied by 12 to obtain a total number of 8502 images in the augmented dataset, including the 654 original images.

4. Implementation

This study performed two scenarios under the same condition, using an Nvidia GTX 1080 TiGPU to train and evaluate eleven PyTorch vision models in a sequential fashion, namely AlexNet, VGG-16, Inception-V1 (GoogLeNet), ResNet-18, SqueezeNet, DenseNet, ResNext, MobileNet, Wide ResNet, ShuffleNet-V2, and MnasNet. We re-trained each model on five tasks, namely MNIST, CIFAR10, Hymenoptera, smartphones, and augmented smartphones, each in a grid search over learning rates $\eta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$ with the ADAM optimizer and a batch size equal to 10. In our plots, we show only the model with the highest accuracy in the overall learning rate. To overcome overfitting, we performed early stopping, so we saved model weights only if the validation accuracy increased. That is, if the validation accuracy decreased, we still used the best model found so far.

We chose to perform two experiments in our paper where a pre-trained model was used to:

- Fine-tune the classifier layer only: This method keeps the feature extraction layers from the pre-trained model fixed, so-called frozen. We then re-initialized the task-specific classifier parts, as given by reference in the PyTorch vision model implementations [42], with random values. If the PyTorch model did not have an explicit classifier part,

for example the ResNet18 architecture, we fine-tuned only the last fully connected layer. We froze all other weights during training. This technique saved training time and, to some degree, overcame the problem of a small-sized target dataset because it only updated a few weights;

- Fine-tune all layers: For this method, we used the PyTorch vision models with original weights as pre-trained on ImageNet and fine-tuned the entire parameter vector. In theory, this technique achieves higher accuracy and generalization, but it requires a longer training time since it is used for initializing weights by continuing the backpropagation instead of random initialization in scratch training.

PyTorch vision models typically have a classifier part and a feature extraction part. Fine-tuning the output layers means fine-tuning the classifier part, which results in a large variation in the model size. We froze all other weights during training. We assessed the model performance with four metrics: the accuracy, the accuracy density, the model size, and training time on a GPU.

4.1. Accuracy Density

This represents the accuracy divided by the number of parameters:

$$density = \frac{accuracy}{\#parameters} \tag{1}$$

A higher value corresponds to a higher model efficiency in terms of parameter usage.

4.2. Accuracy and Model Sizes vs. Training Time

Along with measuring accuracy across tasks, we also measured the training time in seconds and the number of learning parameters in MB. The more complex the model is, the more parameters need to be optimized. When determining the memory utilization of a GPU for each model, the number of parameters is critical. This is the amount of memory that will be allocated to the network and the amount of memory needed to process a batch.

5. Results

We present our results for two experiments, learning from one episode and learning from ten episodes. In each experiment, we tested the fine-tuning of both the classifier batch and the entire network. In the configurations with few shots, each sample was presented only once in a single training episode, while in the configuration with ten episodes, each sample was presented ten times accordingly.

5.1. One-Episode Learning

5.1.1. Fine-Tuning the Full Layers

As shown in Figure 3, we calculated the average accuracy densities of all tested datasets, and we found that SqueezeNet with full tuning showed the highest accuracy density among all models, particularly AlexNet, which came in tenth place. This result affirmed the original hypothesis when SqueezeNet was designed, that it preserves AlexNet's accuracy with 50-times fewer parameters and less than a 0.5 MB model size [33].

5.1.2. Fine-Tuning the Classifier Layers Only

The results, as seen in Figure 3, were slightly different in terms of the accuracy density in the order of the models, but it showed a big difference in the values, where ResNet18 was the most suitable candidate. Each Dataset is tested for both experiments and shown in detail in Appendix A.

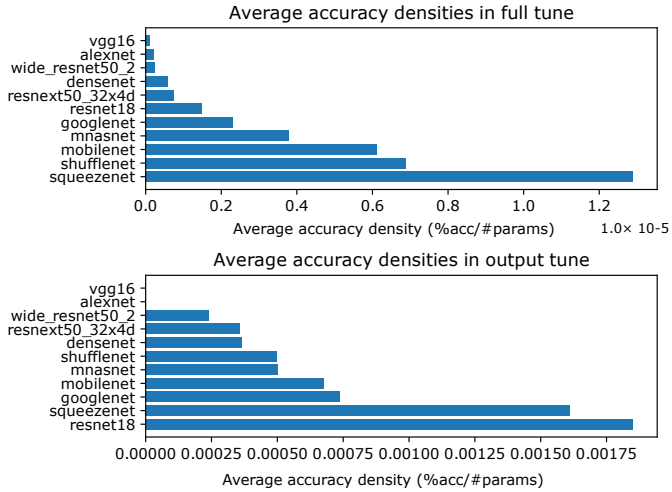

Figure 3. Average accuracy densities with full tuning and tuning the classifier layer only for one episode.

5.2. Ten-Episode Learning

We tested 10 independent trials and calculated the average results to avoid any bias, as follows.

5.2.1. Fine-Tuning the Full Layers

Figure 4 shows that ten-episodes learning did not affect the order of the models in terms of the accuracy densities compared with the one-episode experiments, and the values were higher to a small degree. Figure 5 shows the average model sizes and accuracy vs. the training time for all tasks and models after fine-tuning the full layers of all datasets.

5.2.2. Fine-Tuning the Classifier Layers Only

We found that ResNet18 showed a satisfactory result again as the most efficient model that used its parameters efficiently, as shown in Figure 4. Figure 6 shows the average model sizes and accuracy vs. the training time for all tasks and models after fine-tuning the classifier layer only of all datasets. MnasNet had the poorest performance, making it the least-favorable model in terms of the error metrics, yet it showed a low model complexity and a short training time. The most complex model in all experiments was VGG-16, and the accuracy density figures confirmed this fact. As a result, it might be less trustworthy for embedded and mobile devices. Each Dataset is tested for both experiments and shown in detail in Appendix B.

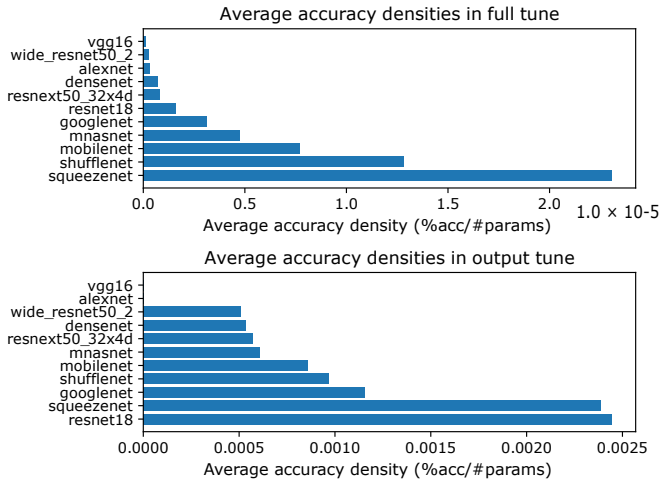

Figure 4. Average accuracy densities with full tuning and tuning the classifier layer only for ten episodes.

Tuning hyperparameters means finding the best set of parameter values for a learning algorithm. In CNNs, the initial layers are designed to extract global features, whereas the later ones are more task-specific. Therefore, when tuning the classification layer, only the final layer for classification is replaced, while the other layers are frozen (the weights of the other layers are fixed). This means utilizing the knowledge of the overall architecture as a feature extractor and using it as a starting point for retraining. Consequently, it achieved high performance with a smaller number of parameters and a shorter training time, as shown in Figure 6. Usually, this scenario is used when the target task labels are scarce [44]. On the other hand, full tuning means retraining the whole network (the weights are updated after each epoch) with a longer training time and more parameters, as shown in Figure 5. When target task labels are plentiful, this scenario is typically applied. Each Dataset is tested for tuning the classifier layer only and tuning full layers and shown in detail for ten episodes in Appendix C, and for one episode in Appendix D.

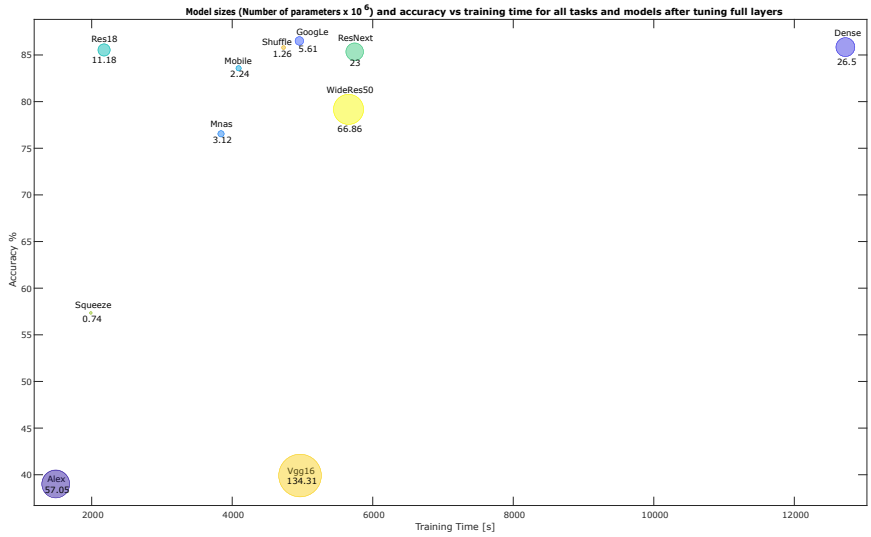

Figure 5. Model sizes and accuracy vs. training time for all tasks and models after fine-tuning full layers.

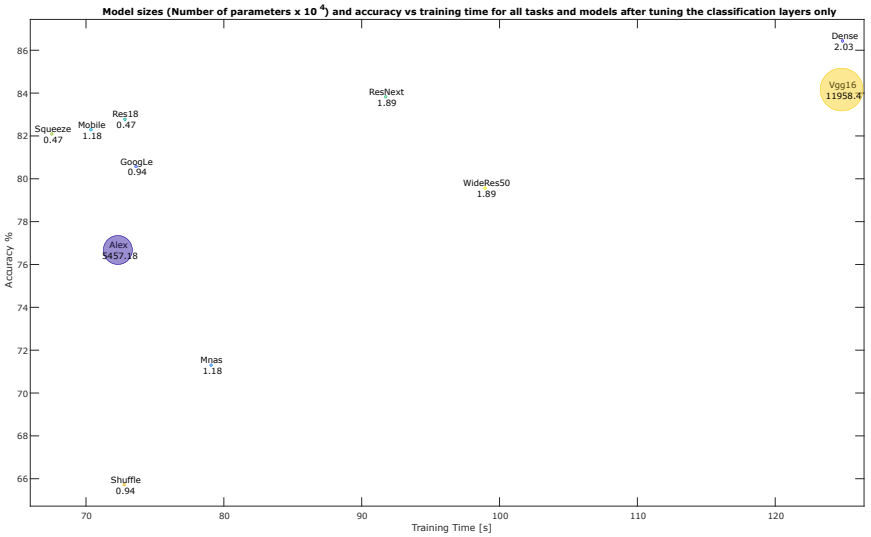

Figure 6. Model sizes and accuracy vs. training time for all tasks and models after fine-tuning the classifier layer only.

6. Conclusions

DNNs' performance has been enhanced over time in many aspects. Nonetheless, there are critical parameters that define which pre-trained model perfectly matches the application requirements. In this paper, we presented a comprehensive evaluation of eleven popular pre-trained models on five datasets as a guiding tool for choosing the appropriate model before deployment. We conducted two different sets of experiments: one-episode learning and ten-episode learning, with each experiment involving tuning the classifier layer only and full tuning. The previous findings, however, might provide some clues for choosing the right model for fine-tuning the classification layer only. For applications that require high accuracy, GoogLeNet, DenseNet, ShuffleNet-V2, ResNet-18, and ResNext are the best candidates, while SqueezeNet is for the accuracy density, and AlexNet for the shortest training time, and SqueezeNet, ShuffleNet, MobileNet, MnasNet, and GoogLeNet are almost equal regarding the smallest model size, for embedded systems applications, for example. On the other hand, we can also provide some suggestions when fine-tuning only the classification layers. DenseNet achieved the highest accuracy, while ResNet18 the best accuracy density, and SqueezeNet the shortest training time. In addition, all models had small model sizes except AlexNet and VGG-16. Although we provided guidelines and some hints, our argumentation does not give a final verdict, but it supports decisions for choosing the right pre-trained model based on the task requirements.

Thus, for specific application constraints, selecting the right pre-trained model can be challenging due to the tradeoffs among training time, model size, and accuracy as decision factors to produce better scores.

For future work, we plan to test more evaluation metrics with the provided parameters to facilitate decision-making in choosing the optimum model to fine-tune. Furthermore, we aim to systematically investigate the usability of all available a priori and a posteriori metadata for estimating useful transfer learning hyperparameters.

Author Contributions: Conceptualization, N.A.B. and N.Z.; methodology, N.A.B.; software, N.Z.; writing—original draft preparation, N.A.B.; writing—review and editing, N.A.B. and N.Z.; supervision, U.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Ministry of Economy, Innovation, Digitization, and Energy of the State of North Rhine-Westphalia within the project Prosperkolleg.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

Appendix A. Accuracy Densities for Each Task with One-Episode Learning

- Accuracy densities for one-episode learning on CIFAR-10, Figure A1;
- Accuracy densities for one-episode learning on Hymenoptera, Figure A2;
- Accuracy densities for one-episode learning on MNIST, Figure A3;
- Accuracy densities for one-episode learning on augmented smartphone data, Figure A4;
- Accuracy densities for one-episode learning on original smartphone data, Figure A5.

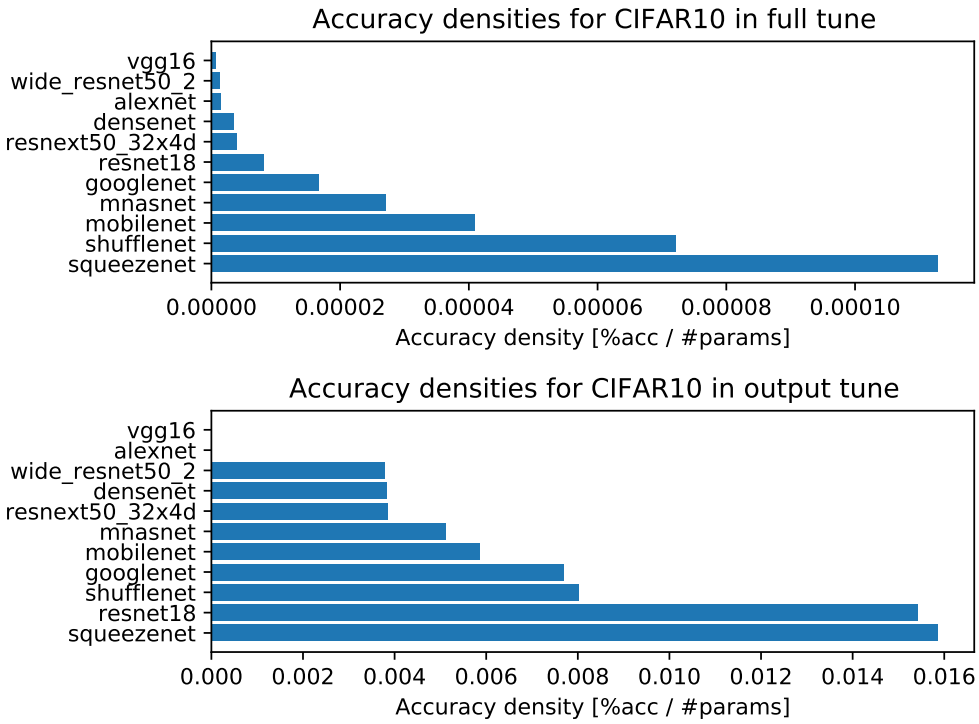

Figure A1. Accuracy densities for one-episode learning on CIFAR-10.

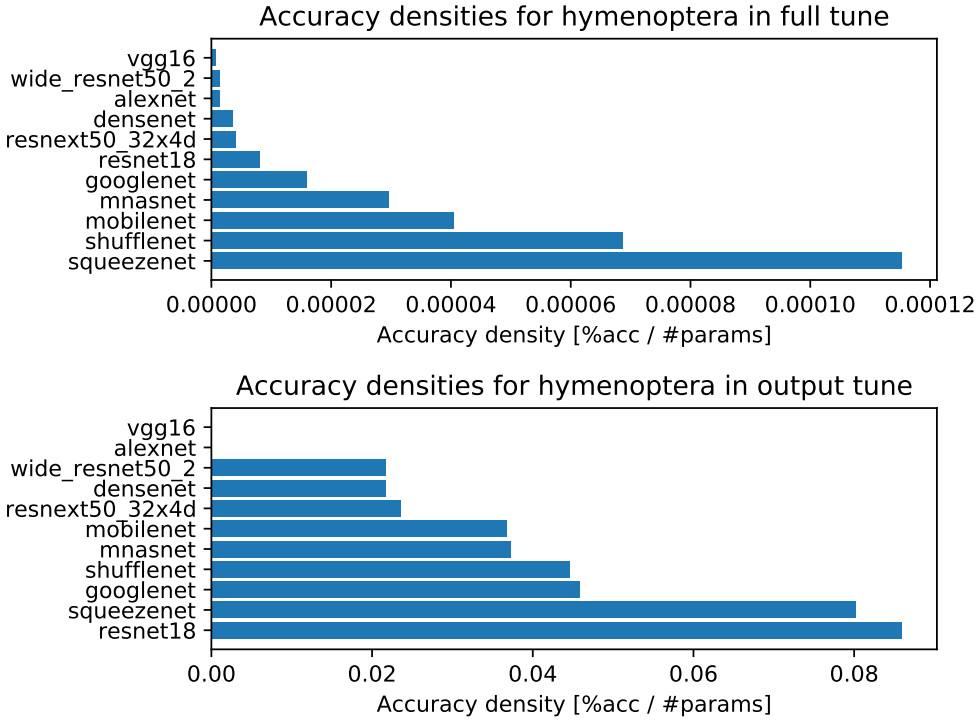

Figure A2. Accuracy densities for one-episode learning on Hymenoptera.

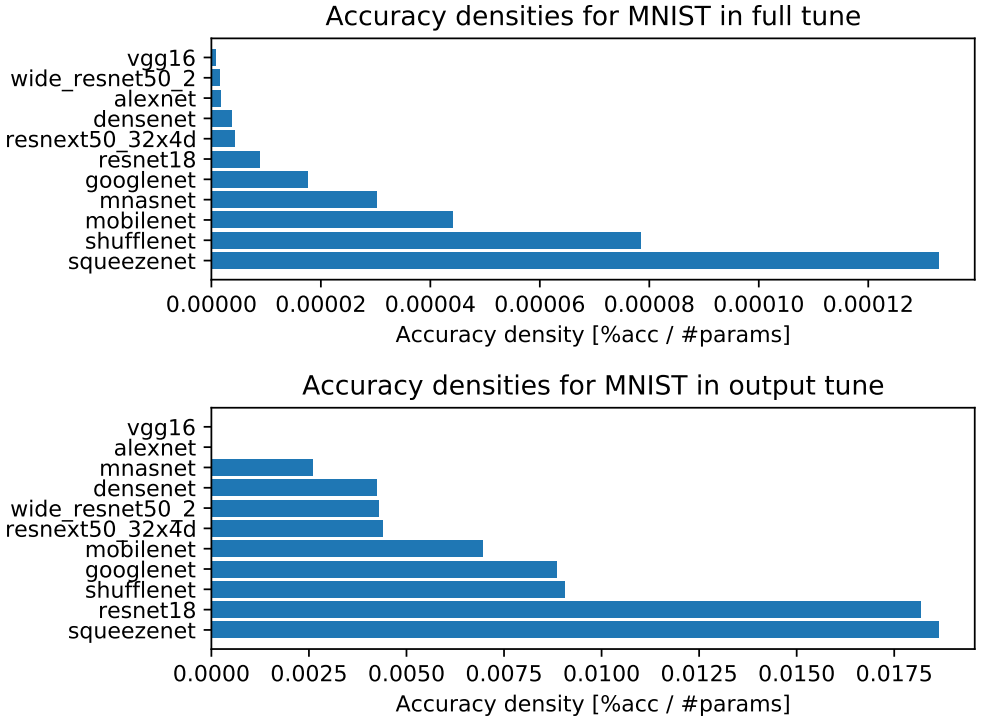

Figure A3. Accuracy densities for one-episode learning on MNIST.

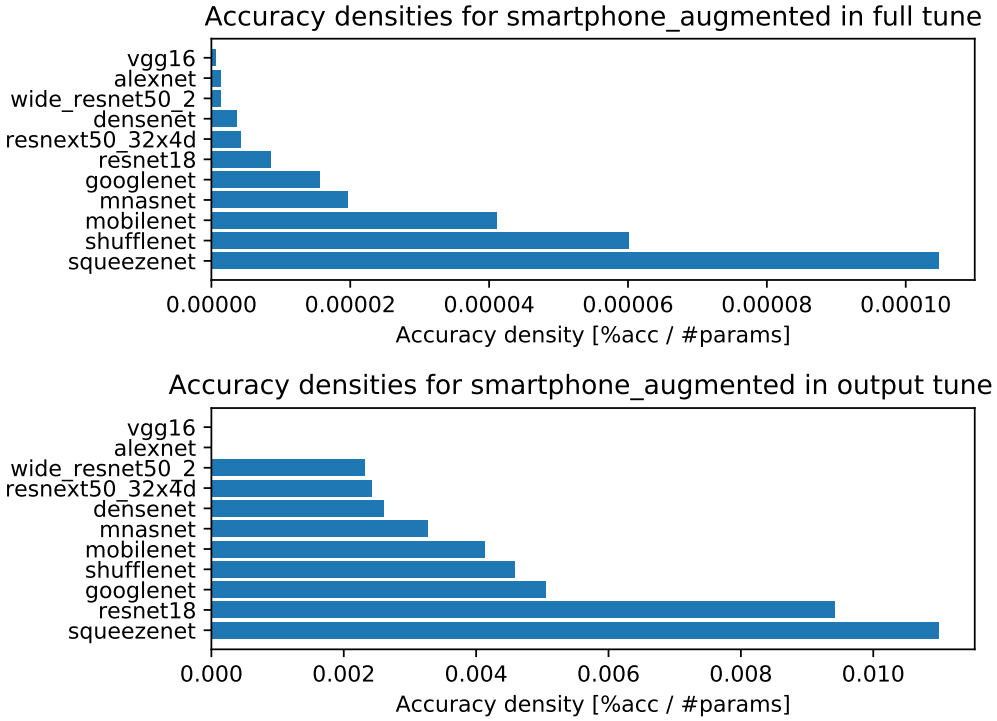

Figure A4. Accuracy densities for one-episode learning on augmented smartphone data.

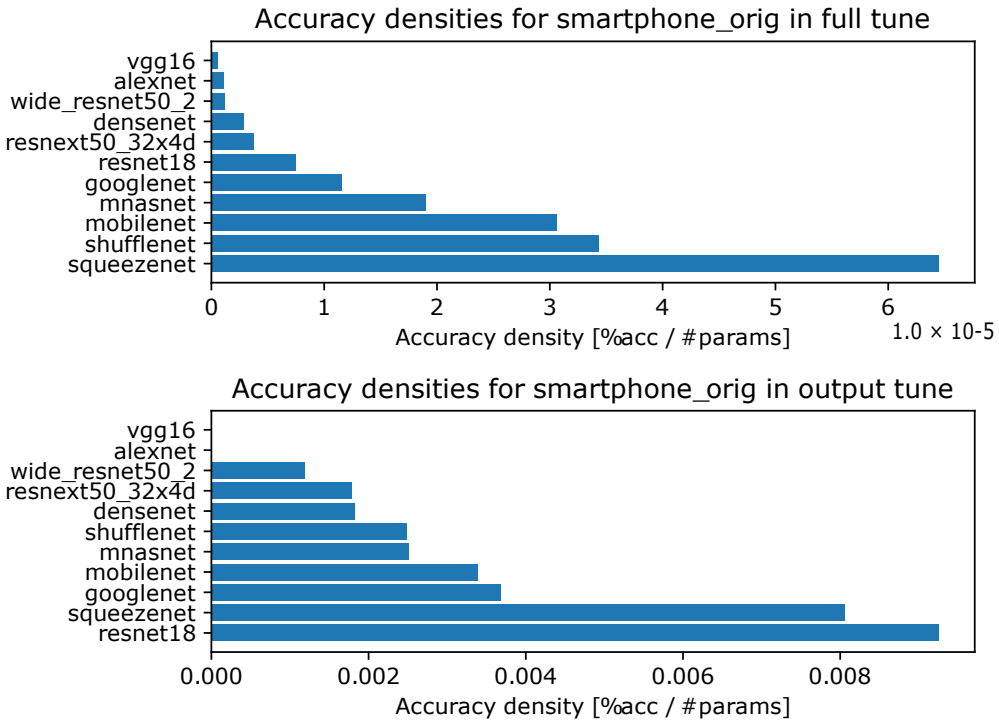

Figure A5. Accuracy densities for one-episode learning on original smartphone data.

Appendix B. Accuracy Densities for Each Task with Ten-Episode Learning

- Accuracy densities for ten-episode learning on CIFAR-10, Figure A6;
- Accuracy densities for ten-episode learning on Hymenoptera, Figure A7;
- Accuracy densities for ten-episode learning on MNIST, Figure A8;
- Accuracy densities for ten-episode learning on augmented smartphone data, Figure A9;
- Accuracy densities for ten-episode learning on original smartphone data, Figure A10.

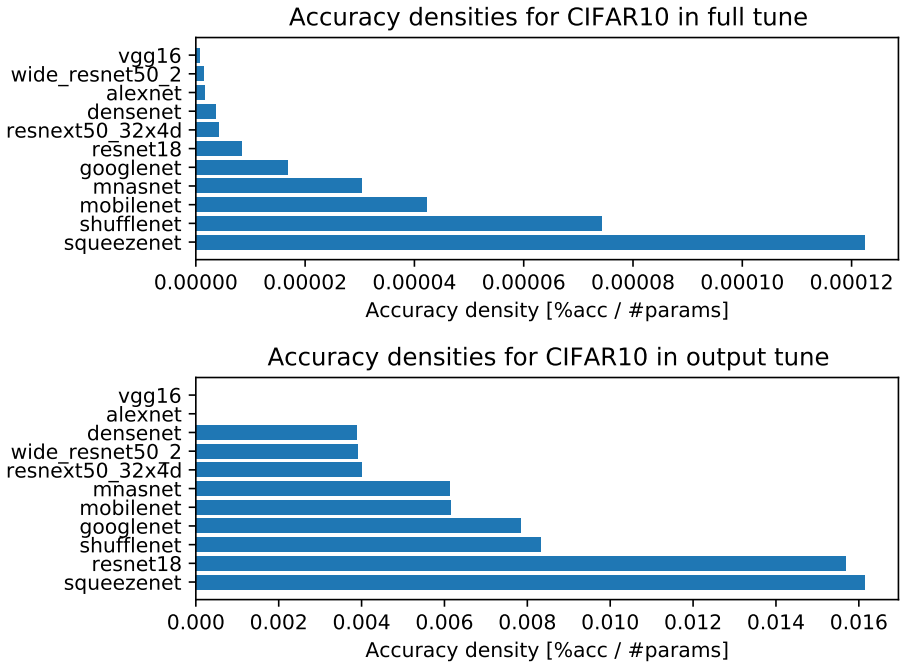

Figure A6. Accuracy densities for ten-episodes learning on CIFAR-10.

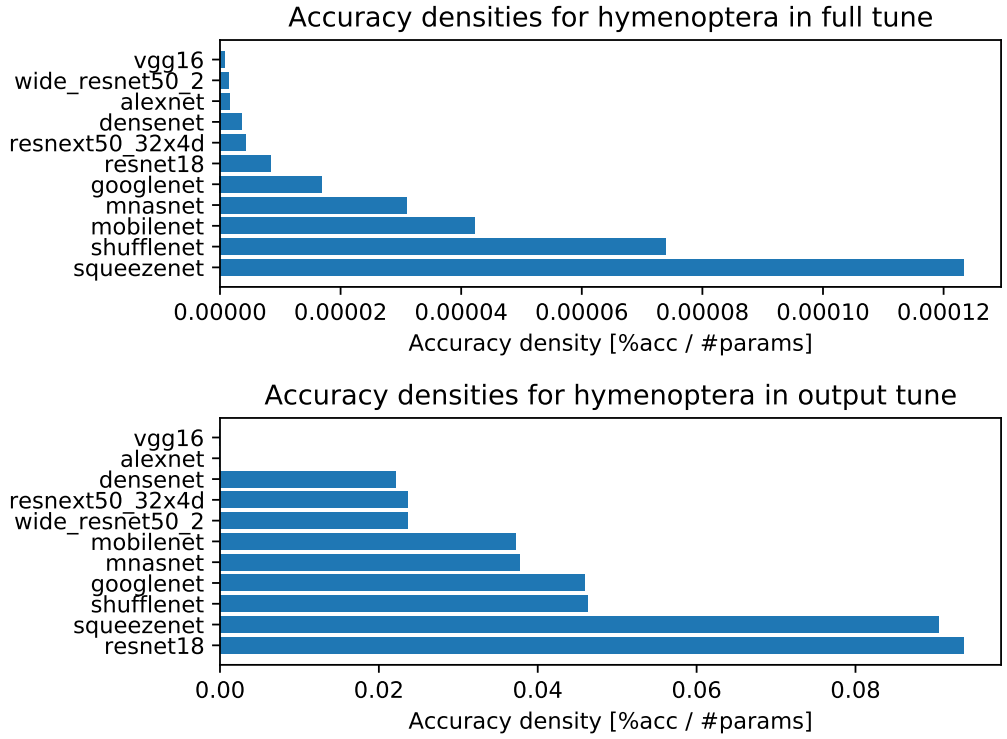

Figure A7. Accuracy densities for ten-episodes learning on Hymenoptera.

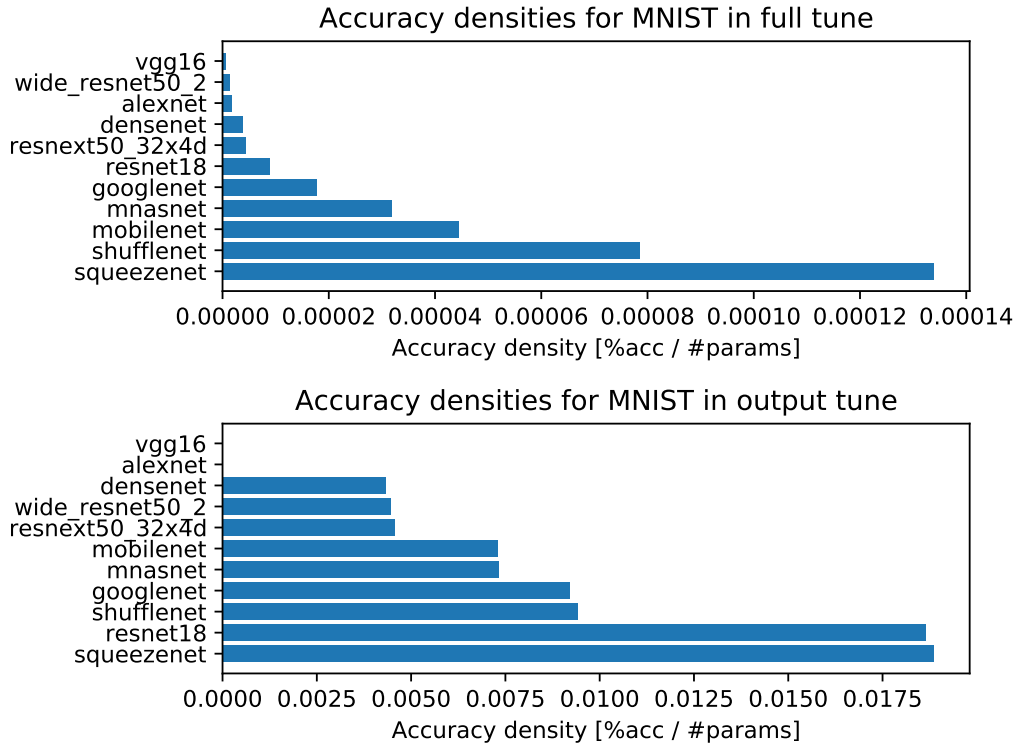

Figure A8. Accuracy densities for ten-episodes learning on MNIST.

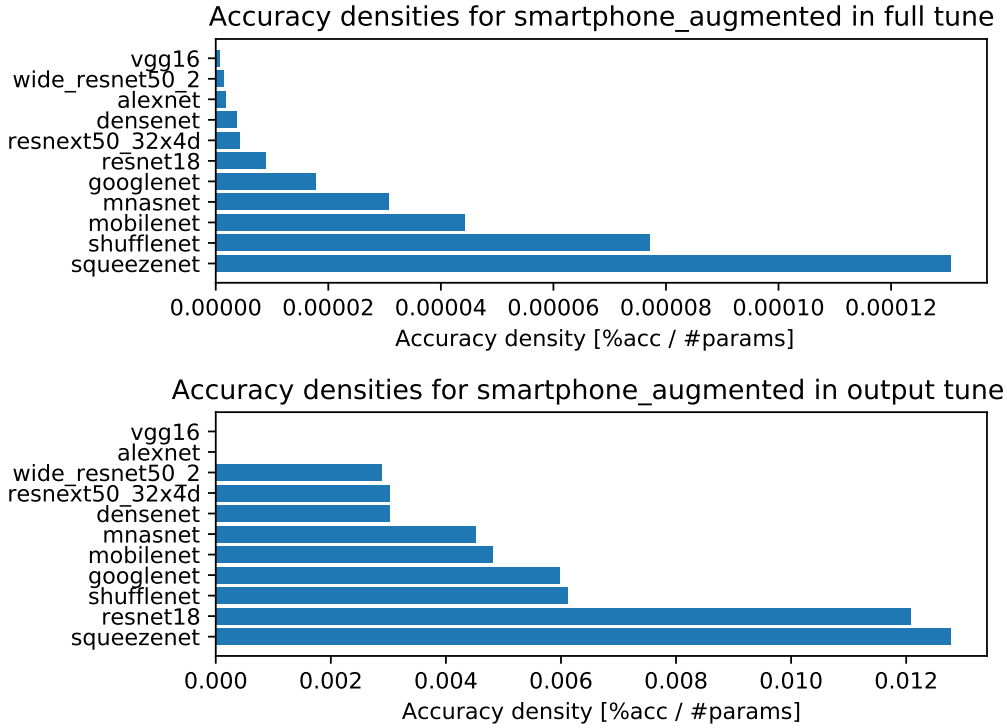

Figure A9. Accuracy densities for ten-episodes learning on augmented smartphone data.

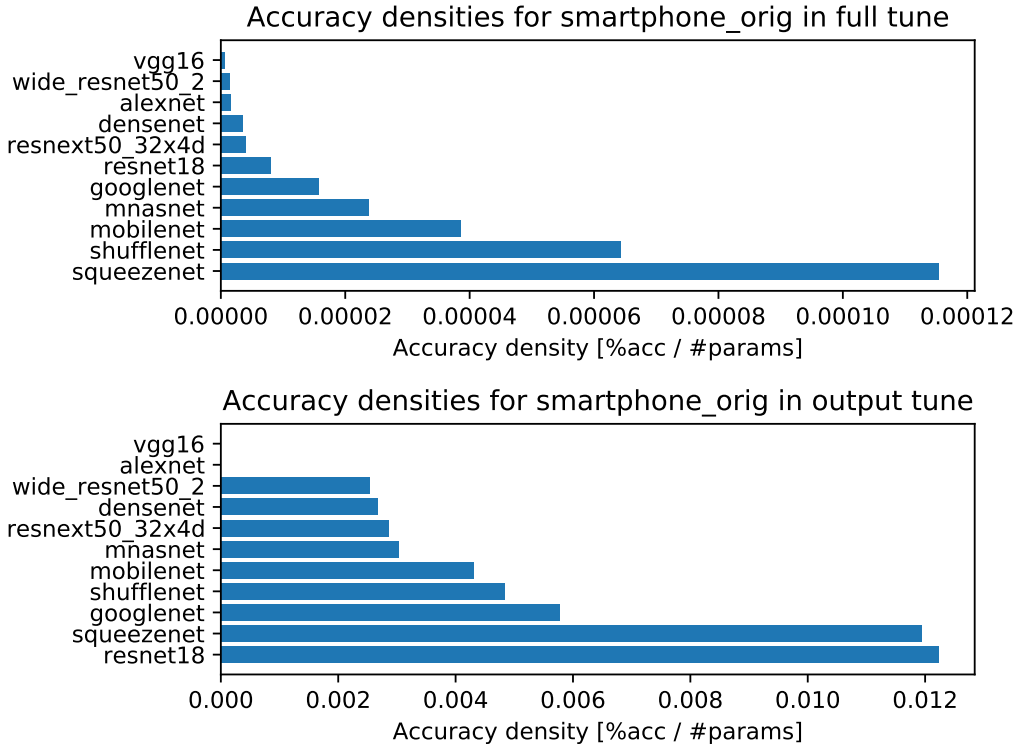

Figure A10. Accuracy densities for ten-episodes learning on original smartphone data.

Appendix C. Accuracy vs. Training Time and Number of Parameters (Model Size) for Each Task with Ten-Episode Learning

- Accuracy vs. training time and model size for CIFAR-10 for ten-episode training, Figure A11;

- Accuracy vs. training time and model size for MNIST for ten-episode training, Figure A12;
- Accuracy vs. training time and model size for Hymenoptera for ten-episode training, Figure A13;
- Accuracy vs. training time and model size for original smartphone data for ten-episode training, Figure A14;
- Accuracy vs. training time and model size for augmented smartphone data for ten-episode training, Figure A15;
- Model sizes and accuracy vs. training time for all tasks and models after fine-tuning the classifier layer only, where A refers to Augmented smartphones, C to CIFAR10, H to Hymenoptera, M to MNIST, and O to the Original smartphone dataset, Figure A16;
- Model sizes and accuracy vs. training time for all tasks and models after full fine-tuning, where A refers to Augmented smartphones, C to CIFAR10, H to Hymenoptera, M to MNIST, and O to the Original smartphone dataset, Figure A17.

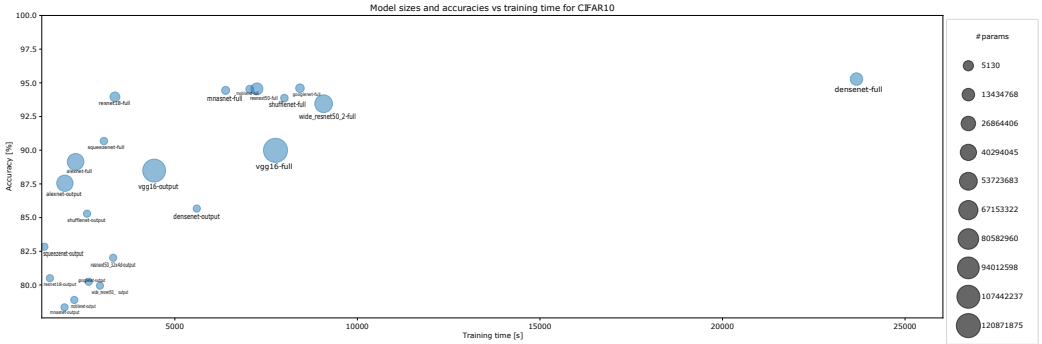

Figure A11. Accuracy vs. training time and model size for CIFAR-10 for ten-episode training.

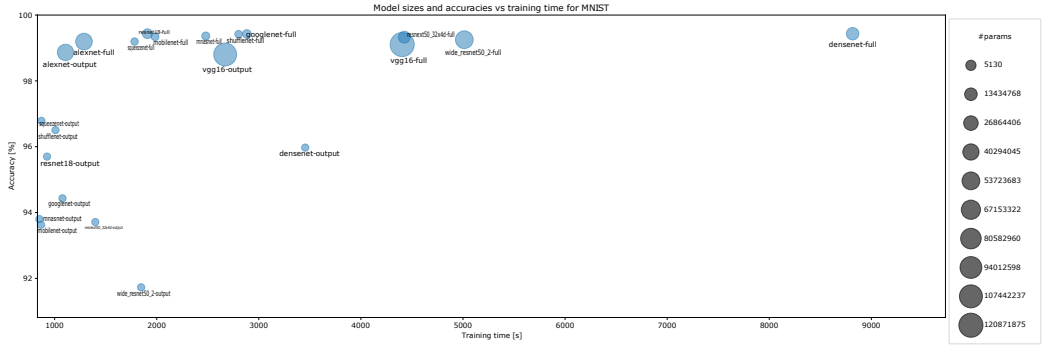

Figure A12. Accuracy vs. training time and model size for MNIST for ten-episode training.

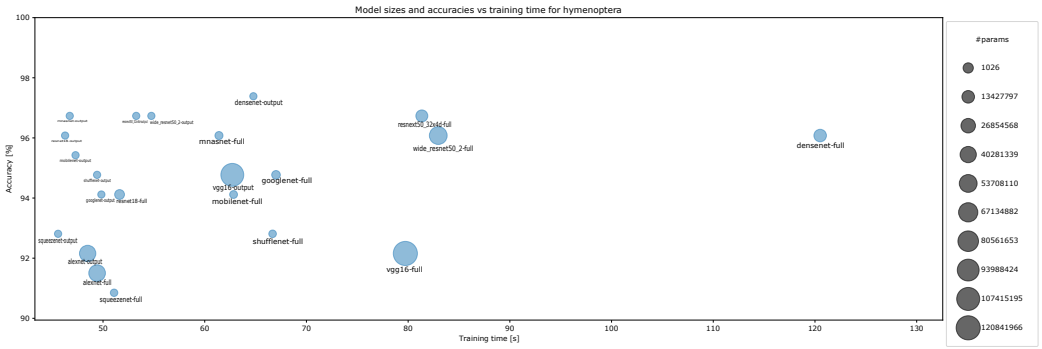

Figure A13. Accuracy vs. training time and model size for Hymenoptera for ten-episode training.

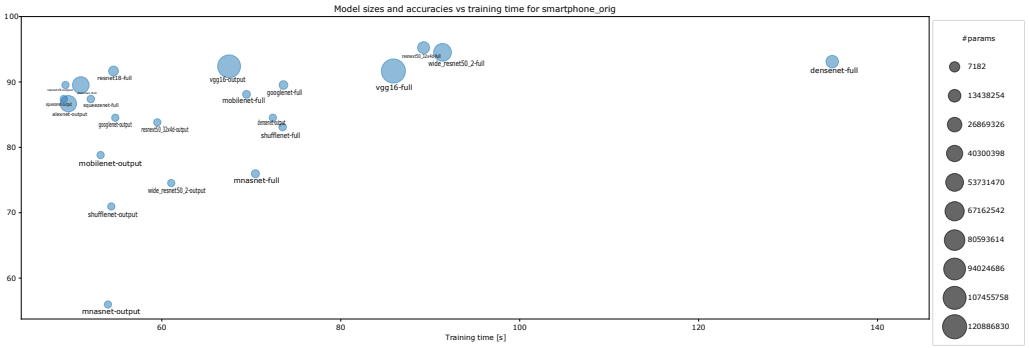

Figure A14. Accuracy vs. training time and model size for original smartphone data for ten-episode training.

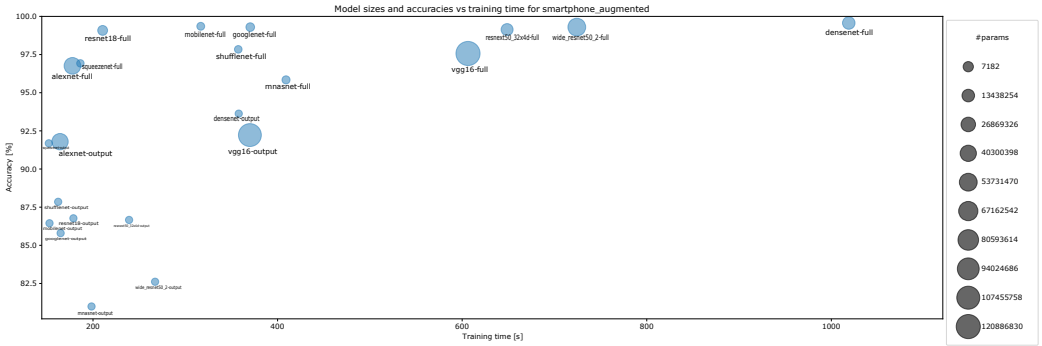

Figure A15. Accuracy vs. training time and model size for augmented smartphone data for ten-episode training.

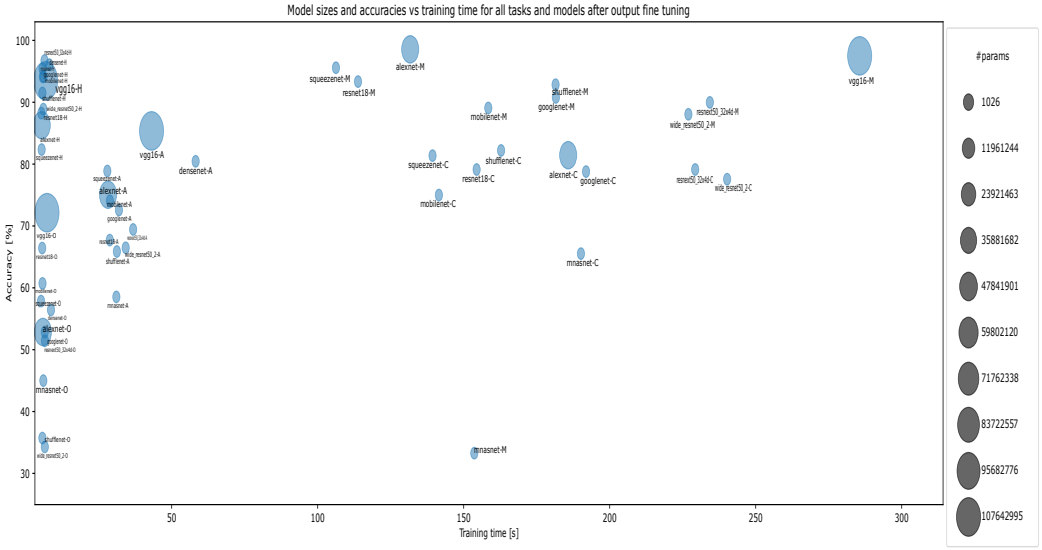

Figure A16. Model sizes and accuracy vs. training time for all tasks and models after fine-tuning classifier layer only, where A refers to Augmented smartphones, C to CIFAR10, H to Hymenoptera, M to MNIST, and O to the Original smartphone dataset.

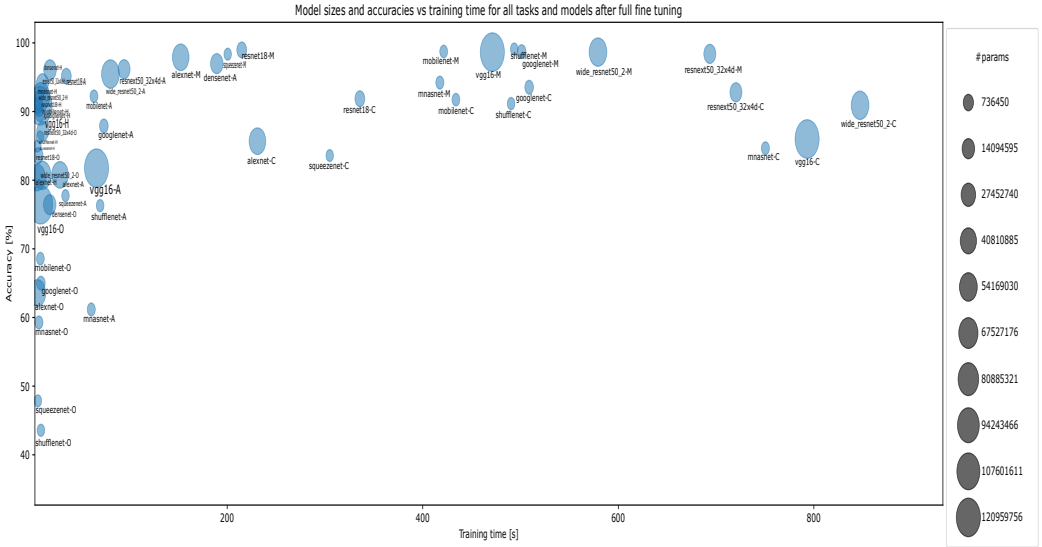

Figure A17. Model sizes and accuracy vs. training time for all tasks and models after full fine-tuning, where A refers to Augmented smartphones, C to CIFAR10, H to Hymenoptera, M to MNIST, and O to the Original smartphone dataset.

Appendix D. Accuracy vs. Training Time and Model Size for Each Task with One-Episode Learning

- Accuracy vs. training time and model size for CIFAR-10 for one-episode training, Figure A18;
- Accuracy vs. training time and model size for MNIST for one-episode training, Figure A19;
- Accuracy vs. training time and model size for Hymenoptera for one-episode training, Figure A20;
- Accuracy vs. training time and model size for original smartphone data for one-episode training, Figure A21;
- Accuracy vs. training time and model size for augmented smartphone data for one-episode training, Figure A22.

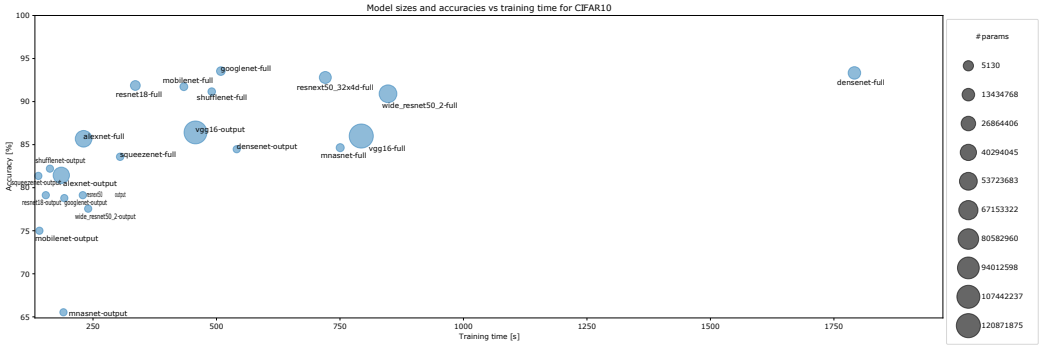

Figure A18. Accuracy vs, training time and model size for CIFAR-10 for one-episode training.

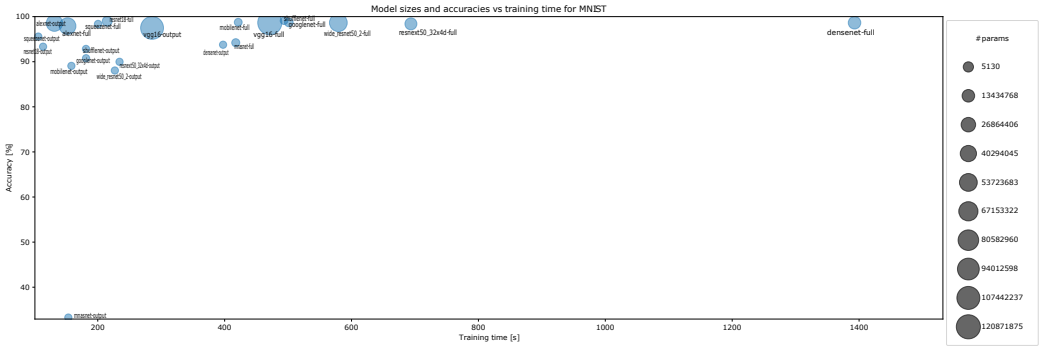

Figure A19. Accuracy vs, training time and model size for MNIST for one-episode training.

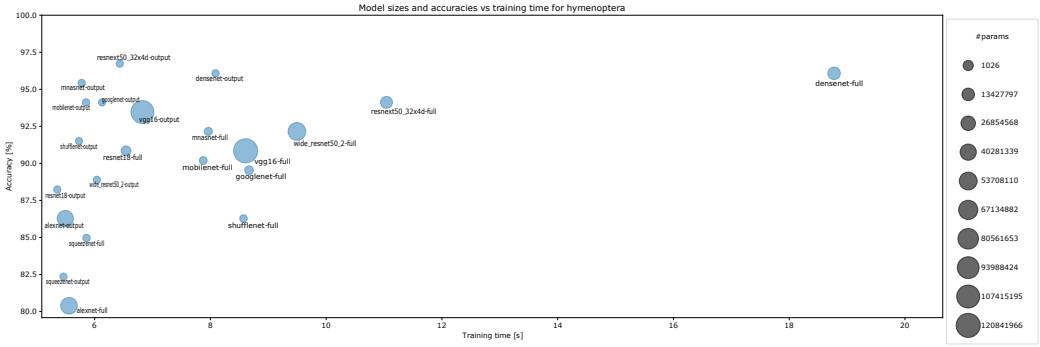

Figure A20. Accuracy vs, training time and model size for Hymenoptera for one-episode training.

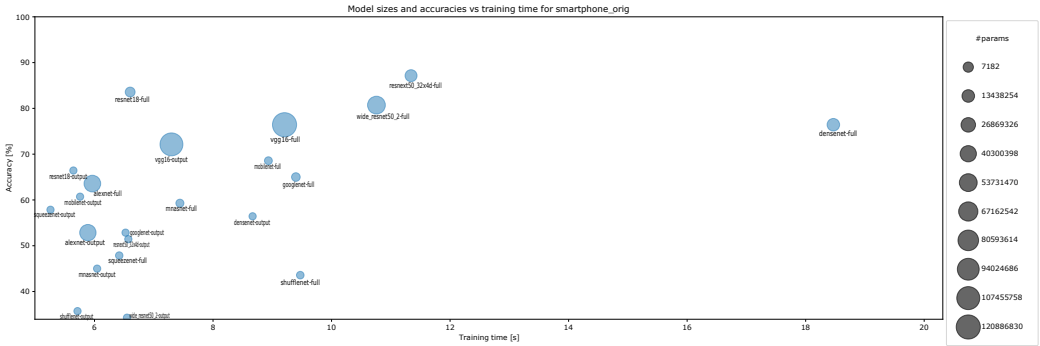

Figure A21. Accuracy vs, training time and model size for original smartphone data for one-episode training.

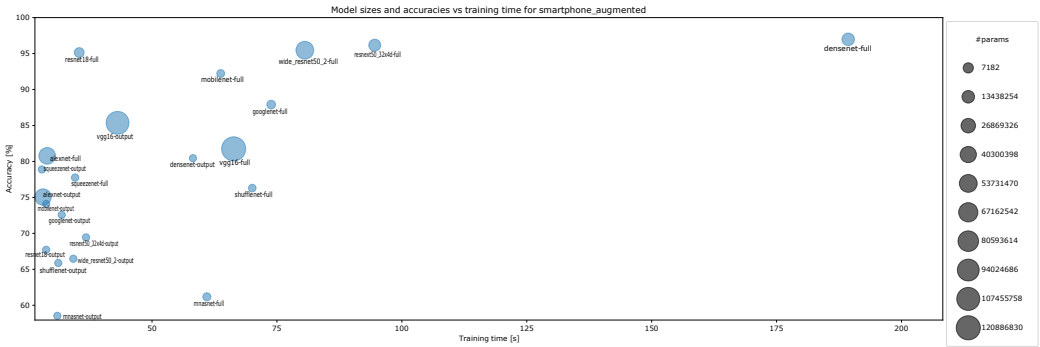

Figure A22. Accuracy vs, training time and model size for augmented smartphone data for one-episode training.

References

1. Lundervold, A.S.; Lundervold, A. An overview of deep learning in medical imaging focusing on MRI. *Z. Fur Med. Phys.* **2019**, *29*, 102–127. [CrossRef] [PubMed]
2. Pires de Lima, R.; Marfurt, K. Convolutional Neural Network for Remote-Sensing Scene Classification: Transfer Learning Analysis. *Remote Sens.* **2020**, *12*, 86. [CrossRef]
3. Zou, M.; Zhong, Y. Transfer Learning for Classification of Optical Satellite Image. *Sens. Imaging* **2018**, *19*, 6. [CrossRef]
4. Abou Baker, N.; Szabo-Müller, P.; Handmann, U. Feature-fusion transfer learning method as a basis to support automated smartphone recycling in a circular smart city. In Proceedings of the EAI S-CUBE 2020—11th EAI International Conference on Sensor Systems and Software, Aalborg, Denmark, 10–11 December 2020.
5. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; de Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-Efficient Transfer Learning for NLP. *arXiv* **2019**, arXiv:1902.00751.
6. Choe, D.; Choi, E.; Kim, D.K. The Real-Time Mobile Application for Classifying of Endangered Parrot Species Using the CNN Models Based on Transfer Learning. *Mob. Inf. Syst.* **2020**, *2020*, 1–13. [CrossRef]
7. Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Transfer learning for time series classification. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018. [CrossRef]
8. Canziani, A.; Paszke, A.; Culurciello, E. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv* **2017**, arXiv:1605.07678.
9. Bianco, S.; Cadene, R.; Celona, L.; Napoletano, P. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access* **2018**, *6*, 64270–64277. [CrossRef]
10. Socher, R.; Ganjoo, M.; Sridhar, H.; Bastani, O.; Manning, C.D.; Ng, A.Y. Zero-Shot Learning Through Cross-Modal Transfer. *arXiv* **2013**, arXiv:1301.3666.
11. Xian, Y.; Schiele, B.; Akata, Z. Zero-Shot Learning—The Good, the Bad and the Ugly. *arXiv* **2020**, arXiv:1703.04394.
12. Lampert, C.H.; Nickisch, H.; Harmeling, S. Attribute-Based Classification for Zero-Shot Visual Object Categorization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *36*, 453–465. [CrossRef]
13. Zhang, Z.; Saligrama, V. Zero-Shot Learning via Semantic Similarity Embedding. *arXiv* **2015**, arXiv:1509.04767.
14. Akata, Z.; Perronnin, F.; Harchaoui, Z.; Schmid, C. Label-Embedding for Image Classification. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *38*, 1425–1438. [CrossRef] [PubMed]
15. Bart, E.; Ullman, S. Cross-generalization: Learning novel classes from a single example by feature replacement. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–26 June 2005; Volume 1, pp. 672–679. [CrossRef]
16. Fink, M. Object Classification from a Single Example Utilizing Class Relevance Metrics. In *Advances in Neural Information Processing Systems*; Saul, L., Weiss, Y., Bottou, L., Eds.; MIT Press: Cambridge, MA, USA, 2005; Volume 17.
17. Tommasi, T.; Caputo, B. The More You Know, the Less You Learn: From Knowledge Transfer to One-shot Learning of Object Categories. In Proceedings of the BMVC, London, UK, 7–10 September 2009. Available online: http://www.bmva.org/bmvc/2009/Papers/Paper353/Paper353.html (accessed on 30 November 2021).
18. Wang, Y.; Yao, Q.; Kwok, J.T.; Ni, L.M. Generalizing from a Few Examples: A Survey on Few-Shot Learning. *ACM Comput. Surv.* **2020**, *53*, 63. [CrossRef]
19. Azadi, S.; Fisher, M.; Kim, V.; Wang, Z.; Shechtman, E.; Darrell, T. Multi-Content GAN for Few-Shot Font Style Transfer. *arXiv* **2017**, arXiv:1712.00516.
20. Liu, B.; Wang, X.; Dixit, M.; Kwitt, R.; Vasconcelos, N. Feature Space Transfer for Data Augmentation. *arXiv* **2019**, arXiv:1801.04356.
21. Luo, Z.; Zou, Y.; Hoffman, J.; Fei-Fei, L. Label Efficient Learning of Transferable Representations across Domains and Tasks. *arXiv* **2017**, arXiv:1712.00123.
22. Tan, W.C.; Chen, I.M.; Pantazis, D.; Pan, S.J. Transfer Learning with PipNet: For Automated Visual Analysis of Piping Design. In Proceedings of the 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, Germany, 20–24 August 2018; pp. 1296–1301. [CrossRef]
23. Montúfar, G.; Pascanu, R.; Cho, K.; Bengio, Y. On the Number of Linear Regions of Deep Neural Networks. *arXiv* **2014**, arXiv:1402.1869.
24. Kawaguchi, K.; Huang, J.; Kaelbling, L.P. Effect of Depth and Width on Local Minima in Deep Learning. *Neural Comput.* **2019**, *31*, 1462–1498. [CrossRef]
25. Khan, A.; Sohail, A.; Zahoora, U.; Qureshi, A.S. A survey of the recent architectures of deep convolutional neural networks. *Artif. Intell. Rev.* **2020**, *53*, 5455–5516. [CrossRef]
26. Hochreiter, S. The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* **1998**, *6*, 107–116. [CrossRef]
27. Srivastava, R.K.; Greff, K.; Schmidhuber, J. Highway Networks. *arXiv* **2015**, arXiv:1505.00387.
28. Hu, J.; Shen, L.; Sun, G. Squeeze-and-Excitation Networks. In Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 7132–7141. [CrossRef]
29. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
30. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv* **2014**, arXiv:1409.1556.

31. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9. [CrossRef]
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. *arXiv* **2015**, arXiv:1512.03385.
33. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv* **2016**, arXiv:1602.07360.
34. Xie, S.; Girshick, R.; Dollar, P.; Tu, Z.; He, K. Aggregated Residual Transformations for Deep Neural Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 5987–5995. [CrossRef]
35. Huang, G.; Liu, Z.; van der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269. [CrossRef]
36. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv* **2017**, arXiv:1704.04861.
37. Zagoruyko, S.; Komodakis, N. Wide Residual Networks. *arXiv* **2017**, arXiv:1605.07146.
38. Zhang, X.; Zhou, X.; Lin, M.; Sun, J. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv* **2017**, arXiv:1707.01083.
39. Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; Le, Q.V. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv* **2019**, arXiv:1807.11626
40. Zaheer, R.; Shaziya, H. A Study of the Optimization Algorithms in Deep Learning. In Proceedings of the 2019 Third International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 10–11 January 2019; pp. 536–539. [CrossRef]
41. Kaziha, O.; Bonny, T. A Comparison of Quantized Convolutional and LSTM Recurrent Neural Network Models Using MNIST. In Proceedings of the 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 19–21 November 2019; pp. 1–5. [CrossRef]
42. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8024–8035.
43. Baker, N.A.; Szabo-Mýller, P.; Handmann, U. Transfer learning-based method for automated e-waste recycling in smart cities. *EAI Endorsed Trans. Smart Cities* **2021**, *5*. [CrossRef]
44. Chen, L.; Li, S.; Bai, Q.; Yang, J.; Jiang, S.; Miao, Y. Review of Image Classification Algorithms Based on Convolutional Neural Networks. *Remote Sens.* **2021**, *13*, 4712. [CrossRef]