Winter 12-2019

Is Bigger Always Better? Lessons Learnt from the Evolution of Deep Learning Architectures for Image Classification

Kai Heinrich

Björn Möller

Christian Janiesch

Patrick Zschech

Is Bigger Always Better? Lessons Learnt from the Evolution of Deep Learning Architectures for Image Classification

Completed Research Paper

Kai Heinrich
TU-Dresden
kai.heinrich@tu-dresden.de
Björn Möller
TU-Dresden
bjoernmdot@web.de
Christian Janiesch
TU-Dresden
christian.janiesch@tu-dresden.de
Patrick Zschech
TU-Dresden
patrick.zschech@tu-dresden.de

Abstract

There exist numerous scientific contributions to the design of deep learning networks. However, using the right architecture that is suited for a given business problem with all constraints such as memory and inference time requirements can be cumbersome. We reflect on the evolution of the state-of-the-art of architectures for convolutional neural networks (CNN) for the case of image classification. We compare architectures regarding classification results, model size, and inference time to discuss the choices of designs for CNN architectures. To maintain scientific comprehensibility, the established ILSVRC benchmark is used as a basis for model selection and benchmark data. The quantitative comparison shows that while the model size and the required inference time correlate with result accuracy across all architectures, there are major trade-offs between those factors. The qualitative analysis further depicts that published models always build on previous research and adopt improved components in either evolutionary or revolutionary ways. Finally, we discuss design and result improvement during the evolution of CNN architectures. Further, we derive practical implications for designing deep learning networks.

Keywords

Deep Learning Architecture, Convolutional Neural Network, Image Classification, Evaluation, Modeling.

Introduction

Deep Learning Networks (DLN) have gained large popularity in the last years due to their ability to classify image information and to detect objects from image data. Applications in the field of computer vision involving DLN, especially Convolutional Neural Networks (CNNs), include object tracking, pose estimation, action recognition, and object counting (LeCun et al. 2015). Application examples include counting fruit for yield prognosis in vineyards or Parkinson's disease prognosis based on image data (Heinrich, Roth, and Zschech 2019; Pianpanit et al. 2019). This development is fueled by the evolution of computing power that allows for faster training and inference times by using large amounts of RAM and GPUs (Cheng et al. 2018).

Besides Computer Vision, CNNs have been applied in the fields of speech recognition and natural language processing, where they outperformed previous state-of-the-art algorithms based on Hidden Markov models and Gaussian mixture models (Cheng et al. 2018). The requirements for implementing a CNN architecture depend largely on the application domain. The general task can be formulated as minimizing inference times on the one hand while maximizing prediction accuracy on the other hand (Larochelle et al. 2007).

Currently, there exist no guidelines or rules on how to efficiently implement a suitable CNN architecture with regard to a specific domain problem (e.g., fault detection through image classification in manufacturing). This can be a problem as the input-output results can seem somewhat arbitrary (Mishkin et al. 2017). This is partly due to a lack of an overview and an understanding of CNN architectures and their

performance implications (Mody et al. 2018). In addition, DLN are considered black box models so that their results cannot be anticipated entirely and thus there is an increasing need for benchmark results and evaluations (Heinrich, Zschech, Skouti, et al. 2019).

Therefore, as a research goal we aim to provide a state-of-the-art analysis of the evolution of DLN architectures for image classification to derive guidelines from best practice applications. Specifically, we focus on CNNs as the predominant type of network for this classification task. In particular, we aim to provide different evaluation results and metrics to give an overview about *(i)* the evolving characteristics of CNN architectures, *(ii)* their prediction performance, and *(iii)* their computational requirements. Additionally, we focus on a comparison over time to depict the technological trend (or evolution) of CNN design. We use the gained knowledge to formulate five guidelines to improve the methodical choice of design and architectures of CNNs.

The paper is structured as follows: The following section gives some preliminaries on the different types of CNN architectures in the field of image classification. Subsequently, we describe our research method including dataset and evaluation criteria in detail. We then proceed with a presentation of the relevant CNN models considering their architectural components and evaluation results, followed by a comprehensive model comparison based on our evaluation criteria. Thereafter, we discuss the evolution of CNNs and the formulate guidelines for choosing an appropriate design and architecture for a given task. The paper concludes with a summary and an outlook as well as limitations to our body of work in the last section.

Architectural Principles of Convolutional Neural Networks

The architecture of a basic neural network including DLN consists of two major viewpoints: *(A) the layer perspective* and the *(B) connection/ network perspective*. Perspective (A) defines which type of layer is used within a network and how each layer is parameterized. Perspective (B) zooms out and depicts the number of each type of layer and how they are connected (Heinrich and Fleissner 2018; LeCun 1989; LeCun et al. 2015). Together (A) and (B) form the network architecture.

The basic architecture of a CNN is depicted in Figure 1. Concerning the entire network from perspective (B), each CNN consists of several different layers that transform the input data into an output format. It follows the structure of *input layer, feature extraction part*, *classification part*, and *output layer*. For the purpose of image classification, the format for the input layer is usually given by a tensor of dimension $width \times length \times color$, where color refers to the different color pixel values (e.g., red, green, blue for RGB) and the output format is a classification decision (e.g., "boat") (Heinrich, Zschech, Möller, et al. 2019).

The feature extraction part consists of at least one convolutional layer. The convolutional layer acts as filter kernel to extract features from its input, which can be the input layer or another preceding convolutional layer respectively. From perspective (A), each convolutional layer consists of several neurons, of which each represents a filter matrix that is multiplied with the input. It can be interpreted as filtering mechanism to extract specific features (e.g., diagonal edges).

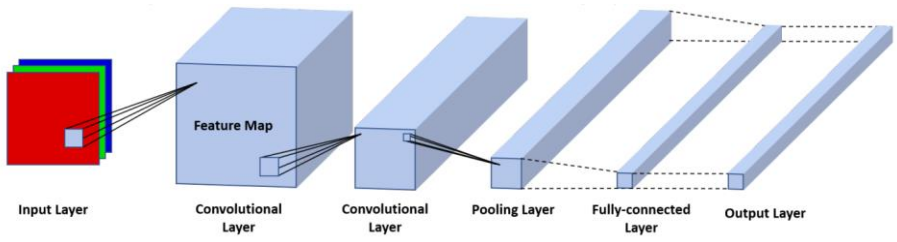
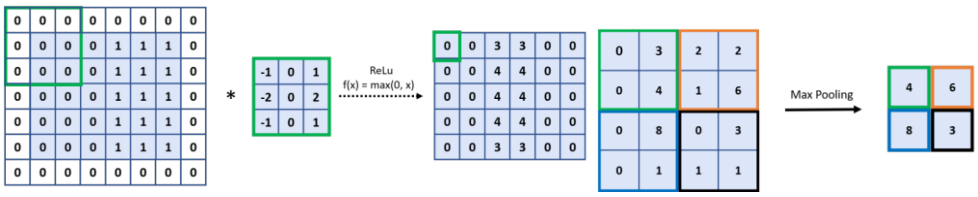


Figure 1. Basic Architecture of a Convolutional Neural Network

The result of the multiplication is saved in a feature map. Figure 2 (a) depicts that multiplication process that is also called convolution. The feature map is constructed by "moving" the filter across the input, where the moving distance is determined by a hyperparameter called *stride*. In order not to loose information from the corners of an image, there is a padding operation that adds artificial pixel around the edges of an image. A typical padding method, the so called zero padding, is depicted in far most left image in Figure 2 (a), where pixel values of zero (white) are added to the original image (blue pixels). Typically, a rectified linear unit (ReLU) function is applied that transforms negative values to zero.



(a) Convolution Operation (b) Max Pooling Operation

Figure 2. Convolutional Neural Network Operations

Concerning image classification from viewpoint (B), the first convolutional layers extract basic shapes like edges or curves, while layers towards the end of the feature selection part represent more complex features that are more likely to resemble the shape of the complex object to be classified. The visualizations in Figure 3 depicts the increasing complexity of feature map outputs.

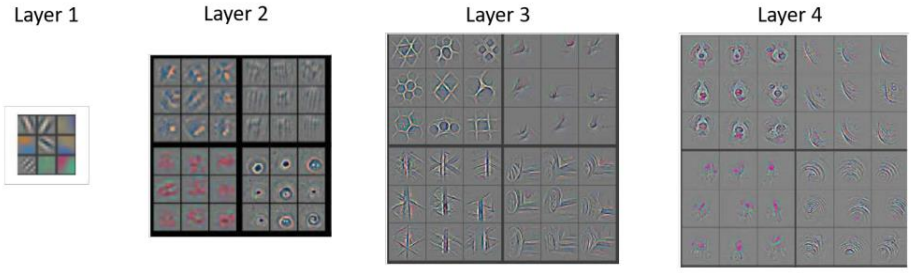


Figure 3. Visualization of Convolutional Layer Activations (Zeiler and Fergus 2013)

After each convolutional layer, the CNN architecture features a pooling layer that reduces the feature map resolution so that the training and inference of a CNN becomes feasible. The pooling layer is parameterized by the degree of reduction (e.g., 4×4 feature map to 2×2 after pooling) and the function to achieve the pooling. A widely used format is the so called max pooling, where parts of the feature maps are compressed by their maximum. This process is depicted in Figure 2 (b). Finally, in the classification part, the fully connected layer flattens out the last resulting feature map of the feature extraction part so that it can be used to generate the results of the output layer. The results are generated using a softmax function that takes the fully connected layer outputs and maps them to confidence values for each class. In addition to these architectural components, in order to prevent overfitting for large networks, one can add dropout layers to forget neurons and make the network more robust and generalizable.

Research Method

Evaluation Dataset

For the purpose of comparing different CNN architectures, we identified a comprehensive collection of network architectures by conducting a review based on submissions to a popular benchmark challenge: the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The challenge was held annually from 2010-2017. The corresponding dataset consists of over 14 million annotated images. The annotations are organized by a semantic network containing synonyms derived from WordNet. Synonym sets were then collected making up to 1000 categories for the classification task. The training set consists of 1.28 million images, where the sample distribution is characterized by 700 to 1300 images per category. The validation and test datasets contain 50,000 and 10,000 images that were uniformly sampled over all categories. The test dataset is not known to the contestants and is used as an evaluation set to determine the winner.

The competition consists of multiple tasks. We focus on the image classification task, where the benchmark results are given by the misclassification rate of the top-1 and top-5 hits. In more detail, the top-5 metric considers an image as correctly classified when the correct category is among the top-5 predictions from the CNN, whereas the top-1 only considers the model's first prediction. Since ambiguity is a big issue in images with multiple objects, the top-1 metric poses a problem if multiple objects are classified correctly. For example, given that an image contains an apple and a lemon but it is annotated with lemon only, it

results in a wrong prediction if apple is the no. 1 prediction and lemon the no. 2 prediction, although the network correctly detected both objects. Our final review result set of models comprises the winners of the ILSVRC image classification task from 2012-2017 since they clearly represent the state-of-the-art evolution in DLN architectures as judged by peer scientists and the given benchmark results.

Evaluation Criteria

We propose an evaluation design that considers three main factors: *(i) model size*, *(ii) classification results*, and *(iii) time of inference* (Larochelle et al. 2007).

The *model size* is evaluated by taking into account the amount of information that is encoded by the model and the memory requirements for model inference. Together, those criteria can be used to evaluate whether a model is able to work in a feasible manner given the hardware restrictions in the form of RAM specifications of the used information system's hardware. Model size is measured by (*i-i*) *number of parameters* that result from training and *(i-ii) number of extracted feature values*. The number of trainable parameters consists of the filter kernel matrix elements as well as the parameters of the fully connected layers together with their bias weights. Assuming a convolutional layer filters every feature map of their respective input layer, the number of parameter for a given convolutional layer j with filter width w_j, height h_j and number of feature maps k_j is given by:

$$Parameters\ (Conv_j) = (w_j h_j k_{j-1} + 1)k_j$$

While there are no trainable parameters in the pooling layers, the fully connected layer takes n inputs and connects them all to m outputs. Together with a bias term, the number of parameters in the fully connected layer can be expressed as follows:

$$Parameters\ (FC) = (n + 1)m$$

The second component, that is the number of feature values, becomes only relevant at runtime during the inference process. It is expressed as the number of stored values during the CNN's runtime. The value can be expressed by the entirety of all feature maps from the convolutional layers and the entirety of all neurons in the fully connected layers. While the number of neurons for the fully connected layer is directly given as a hyperparameter of the architecture, the feature map sizes and frequencies of a layer j depend on the size of the input map $w_{i(j)} \times h_{i(j)}$ for layer j, padding p (e.g., adding zeros around the borders of an image), stride s and filter size $w_j \times h_j$. Assuming quadratic feature maps with $w_j = h_j$ and $w_{i(j)} \times h_{i(j)}$, the number of features of a convolutional layer j with k_j maps can be expressed as:

$$Features\ \left(Conv_j\right) = \left(\frac{w_{i(j)} - w_j + 2p}{s} + 1\right)^2 k_j$$

The model size is usually not given within the selected research papers, so we calculated them based on the given model architectures using the above stated calculation procedures.

The *classification results* are directly taken from the ILSVRC classification task benchmark and measured in terms of prediction accuracy. The measure of accuracy is calculated as the proportion of images in a dataset that were classified correctly with regard to all images. Zeiler and Fergus (2013) showed that using the benchmark dataset, the resulting models can be transferred to similar tasks, which makes the ILSVRC benchmark a representable instance for performance measurement for classification tasks. For comprehensibility and maximum comparability, we exclude model approaches that considered features derived from other data than the image dataset. In addition, we circumvent the problem of ensemble models that usually reflect the best results but are a combinations of similar models, as it biases the evaluation of single architectures, which is the goal of this paper. We therefore also include the single models into our evaluation design, whenever an ensemble model was the best model with regard to the given task.

To measure the *inference time,* we considered the required floating point operations of the observed model while executing one feed forward run. This reflects the time the CNN would take when put into practice applications. As such, this is a very important benchmarking KPI, as there are often constraints on how long object detection may take to complete the prediction (e.g., detection of material flaws in a high-frequent, strictly timed production chain).

Model Architectures

In this section, we present the results of the evaluation of the different CNN architectures. We considered all best performing models from the years 2012-2017. Table 1 contains all architectures sorted by year and the main eight architectural concepts. In order to achieve improved comparability, we provide absolute and relative measures with regard to the evaluation criteria introduced in the previous section. The relative measurements are given as percentage values with regard to the initial baseline architecture called AlexNet. Our comparison base consists of eight different architectures. For the reason of scientific comprehensibility and to avoid very similar evaluations, we present only versions with major differences in architectural design (e.g., number of layers). When the values could not be obtained or could not be reconstructed due to missing information, we marked the table cell with a "-".

Year	Architecture		Model Size		EN	Classification Results		Inference Time
	Model	#Layer	#Features	#Param.		Top-1	Top-5	G-Flops
2012	AlexNet	8 (100%)	932,264 (100%)	60,000,000 (100%)	5	40.70	18.20 (16.42)	720 (100%)
2013	ZF-Net	8 (100%)	2,010,312 (216%)	62,357,608 (104%)	6	37.50	16.00 (14.8)	-
	OverFeat Fast	8 (100%)	1,020,955 (110%)	145,000,000 (242%)	7	38.12	16.27 (14.18)	-
	OverFeat Acc.	9 (112%)	2,404,675 (258%)	144,000,000 (242%)	7	35.74	14.18 (13.6)	-
2014	VGGNet-16	16 (200%)	15,237,608 (1634%)	138,000,000 (230%)	-	24.4	7.2 (-)	15,300 (2125%)
	VGGNet-19	19 (238%)	16,542,184 (1774%)	144,000,000 (240%)	7	24.4	7.1 (7.33)	19,600 (2722%)
	GoogLeNet	22 (275%)	4,028,864 (432%)	6,797,700 (11.3%)	7	-	7.89 (6.67)	1500 (208%)
2016	ResNet-34	34 (425%)	3,915,240 (420%)	21,800,000 (36%)	u	21.53	5.6 (3.57)	3600 (500%)
	ResNet-50	50 (625%)	9,442,280 (1013%)	25,600,00 (43%)	u	20.74	5.25 (3.57)	3800 (528%)
	ResNet-101	101 (1263%)	14,560,232 (1562%)	44,500,000 (74%)	u	19.87	4.60 (3.57)	7600 (1056%)
	ResNet-152	152 (1900%)	20,882,408 (2240%)	60,200,000 (100%)	u	**19.38**	**4.49** (3.57)	11,300 (1569%)
	ResNeXt-50 (32x4d)	50 (625%)	12,195,816 (1308%)	25,600,000 (43%)	u	22.2	- (3.03)	4100 (569%)
	ResNeXt-101 (32x4d)	101 (1263%)	19,019,752 (2040%)	44,500,000 (74%)	u	21.2	- (3.03)	7800 (1083%)
	ResNeXt-101 (64x4d)	101 (1263%)	36,882,408 (3956%)	-	u	20.4	5.3 (3.03)	15,600 (2176%)
2017	SE-ResNet-50	50 (625%)	28,115,456 (3016%)	9,459,936 (16%)	u	23.29	6.62 (2.25)	3870 (538%)
	SE-ResNet-101	101 (1263%)	49,243,680 (5282%)	14,605,088 (24%)	u	22.38	6.07 (2.25)	7600 (1056%)
	SE-ResNeXt-50 (32x4d)	50 (625%)	28,115,456 (3016%)	12,246,816 (20%)	u	21.10	5.49 (2.25)	4250 (590%)
	SE-ResNeXt-101 (32x4d)	101 (1263%)	49,243,680 (5282%)	19,070,752 (32%)	u	20.70	5.01 (**2.25**)	8000 (1111%)

Table 1. Model Evaluation Results

All the information presented in in Table 1 refers to single models. However, if an ensemble version was submitted, the column EN indicates the number of single models combined into an ensemble version of that model type. The classification results in parenthesis in the "Top-5" column give the top-5 misclassification rate of the secret test dataset for the ensemble models (e.g., seven VGGNet-19 models were combined and yield a top-5 test misclassification rate of 7.33 %). If information on the exact composition of an ensemble model was missing, but the results are still known, we indicated it with a "u" in the EN column. In this case, the same known ensemble accuracy is given in parenthesis for all single models of a

model type, since we cannot state the exact composition of the ensemble (e.g., we know there is a ResNet ensemble but we do not know the type and number of models that make up the ensemble model, so we note the same ensemble classification result of 3.57 % in parenthesis for all single ResNet models).

AlexNet

The initial conceptual design of a CNN for image classification faces the problem of a high number of parameters, which leads to severe overfitting. To counter act the overfitting effect, the model AlexNet was therefore additionally trained with some variations of the training inputs to increase robustness and rely on more general features rather than learning enormous amounts of noise (Krizhevsky et al. 2012). Another step towards a more robust model was the introduction of the drop-out mechanism in both fully connected layers, meaning that not all hidden units were used, but rather some sort of regularization is applied to keep only the most important ones. The AlexNet architecture basically consists of 8 layers: 5 convolutional layers with respective pooling layers and 3 fully connected layers. The first C1-layer produces 96 feature maps applying a filter kernel of 11×11×3 with a stride of 4. For feasibility reasons, the kernels were split onto two graphic cards, so that 48 feature maps were generated per GPU. C2 yields 256 feature maps applying a filter of 5×5×48. The feature maps are pooled using maximum pooling and they are normalized. The layers C3-C4 are without pooling and normalization and they work autonomously on separate graphic units. Before applying two fully connected layers F6 and F7 with 4096 units each (=neurons), an additional normalization and pooling is applied. The output layer takes the input from the F7 and applies the softmax function yielding result values for all 1000 classes. The architecture is depicted in Figure 3.

The AlexNet uses the 10-Crop-voting mechanism that was also introduced by Krizhevsky et al. (2012) to vote for the final output classification scores. The cropping procedure involves splitting the image into a central crop and four corner crops. The output vectors of those crops, together with their horizontal flip versions are then averaged to yield the classification vector.

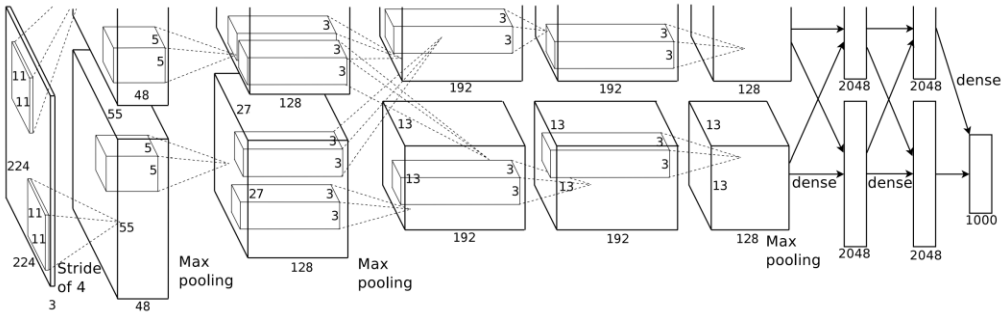


Figure 3. Architecture of AlexNet (Krizhevsky et al. 2012)

ZF-Net

The ZF-Net architecture improved the AlexNet architecture by first diagnosing the features that are learned in higher layers and secondly by using this information to improve the layer architecture. Zeiler and Fergus (2013) found that the previous CNN architectures reacted strongly to the special distribution of features with regard to the input image and therefore they introduced some more granularity in the C1 filter. The main architectural differences are given by the limitation towards one GPU and more detailed filters in the first convolutional layer of 7×7 with a stride of 2 instead of 4, avoiding the anti-aliasing effect in the second layer and therefore achieving better benchmark results in the end at the cost of more feature values to be produced. The ZF-Net also uses the 10-Crop-voting mechanism introduced by Krizhevsky et al. (2012).

OverFeat

The OverFeat architecture tried to overcome the problem that objects on an image are not necessarily central by implementing a sliding window approach which selects multiple sub-images of different sizes that are then used as training inputs. The different sizes of the sub-images require the introduction of a resolution augmentation function after the feature extraction step, before the classifier is applied (Sermanet et al. 2013). OverFeat was introduced with two versions: a fast one and an accurate one. Apart from an

increased number of feature maps and additional pooling, the basic architecture of OverFeat is based on AlexNet with 5 convolutional layers and 3 fully connected layers. The accurate variation adds another convolutional layer in the feature extraction part of the network. To efficiently generate sub-images from a given feature map, only one feature map is generated and then scaled towards six different resolutions. Since more inputs are used, the number of features increased compared to the previously observed architectures. Overfeat uses the random-crop mechanism introduced by Krizhevsky et al. (2012), where in comparison to the 10-crop mechanism the selection of the crops is random.

VGGNet

The intention behind the architecture of VGGNet was to measure the influence of network depth on classification results (Simonyan and Zisserman 2014). Based on the AlexNet architecture with more detailed filters (smaller width, height and strides), the VGGNet approach uses the entire image rather than sub-images (as compared to OverFeat), but more layers are added subsequently to arrive at a 19 layer architecture. The hyperparameters for each architectural module was set beforehand: convolutional filters were set at 3×3 with stride 1 and every layer consists of five blocks of convolutional layers with an equal number of feature maps. Only the depth of the feature maps increases from 64 to 512 from the first to the last convolutional layer. Every network consists of 3 fully connected layers. Models were trained with 11, 13, 16 and 19 layers of which we only considered the variations with 16 and 19 layers in Table 1, since they are the ones that improved classification results. The inference times for the VGGNet-16 and VGGNet-19 models are based on He et al. (2015). The authors also introduce a multi-scale-voting mechanism that in comparison to previous crop mechanisms uses three differently scaled versions of the input image with 50 crops each to output an average as the classification result.

GoogLeNet

The aim of introducing the GoogLeNet was to more efficiently use computational resource within a CNN architecture, both in the process of inference and model training (Szegedy et al. 2015). This allows for a deeper architecture with more features maps while simultaneously reducing computational power needed for model inference. The relevance of this step is underlined by the increasing use of mobile and embedded computing. To achieve this, the architecture is constraint by a given maximum inference time. The reduction in computational power is achieved by introducing the *inception module* as an architectural building block. It can best be described as a network within a network architecture where 1×1 convolutional layer are introduced to achieve a reduction of the feature map size.

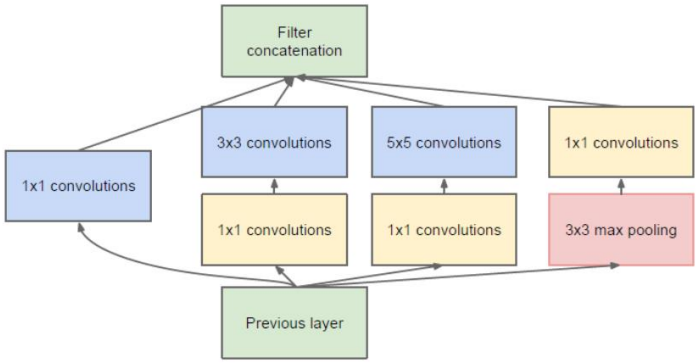


Figure 4. Inception Module of GoogLeNet (Szegedy et al. 2015)

The inception module consist of different convolutional layers with different filters, where the 1×1 convolutional layers are executed before the more computational challenging 3×3 and 5×5 filters. After the initial convolutional layer with 7×7 and stride 2, the input image is reduced to 112×122 with 64 feature maps. The first inception module is applied after layer C3. The inception modules use the same filter sizes but have different feature map resolutions. The GoogLeNet consist of 9 inception modules but does not have the usual 3 fully connected layers. Instead, it applies an average pooling layer that is connected to only one fully connected layer before the softmax activation. For further illustration, Figure 4 depicts the architecture of an inception module. The GooLeNet uses multi-scale-voting with multiple crop sizes of 1, 10

and 144 in comparison to the static 50 of the VGGNet architecture. The submitted winning model uses the 144 version and those results are also given in Table 1 for the single and ensemble model.

ResNet

While the evolution of CNN towards GoogLeNet lead to the assumption that stacking additional convolutional layers would lead to improved classification results, the problem of declining classification results was discovered by He et al. (2015) in experiments that compared an 18 layer CNN with a 34-layer CNN. An important finding from the experiments showed that overlearning was not the reason for the declining classification performance with increasing number of layers. The experiments showed that deeper architectures present both: an increasing training and validation error over all training epochs. Two separate explanations were found for the decline: the vanishing gradient and the degradation problem. The vanishing gradient problem occurs with increasing number of layers and implies that the weights of more advanced layers converge towards zero, especially when using sigmoid activation functions. Due to near zero activations, this problem leads to saturation of the learning effects when using backpropagation. This problem can partially be addressed by normalization of initial weight and normalization layers (Glorot and Bengio 2010). The second problem that occurs when stacking deep architectures is the degradation problem that leads to diminished accuracy values with increasingly deeper architectures. That problem was addressed with the Residual Learning Framework (He et al. 2015). As a solution it was suggested that the convolutional layers are organized in *residual blocks* that use short cut connections to avoid vanishing learning effects through long connection chains.

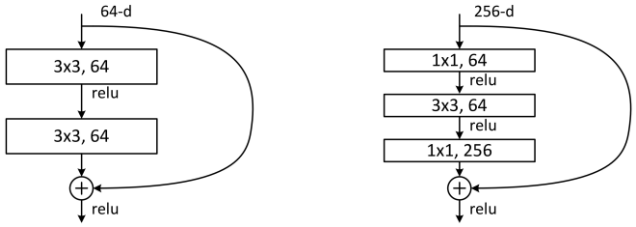


Figure 5. Standard Residual vs. Bottleneck Residual Block (He et al. 2015)

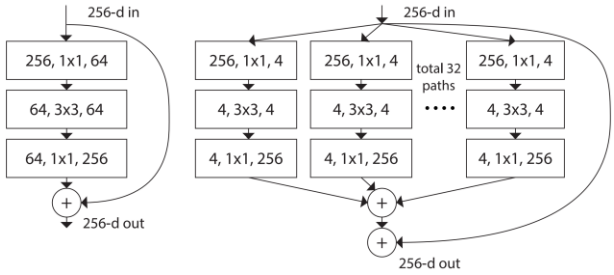


Figure 6. ResNet vs. ResNeXt (Xie et al. 2016)

Applying the architecture reorganization to the 18 and 34-layer networks from the previous experiment, the residual blocks lead to improved accuracy of the 34-layer model compared to the 18 layer model (He et al. 2015). The ResNet provides residual blocks that can skip connections within its convolutional layer chain and directly can be added to the output of the last convolutional layer. They distinguish between a standard residual block containing a chain of 3×3 filter kernels and a bottleneck residual block that consists of a mixture of 1×1 and 3×3 filter kernels. Figure 5 depicts both blocks in comparison. The ResNet uses the 10-Crop-voting mechanism introduced on 5 different input resolutions to generate an output vector.

ResNeXt

Xie et al. (2016) showed that increasing cardinality instead of going even deeper with residual blocks increases accuracy while inference time measured in FLOPS remains the same (Xie et al. 2016). Cardinality refers to the number of parallel paths within a residual block, motivated by the inception modules from GooLeNet. The ResNeXt architecture uses residual blocks but not only in a sequential way within one

bottleneck residual block but in parallel, which increases cardinality. Figure 6 depicts a comparison between the ResNet block and the ResNeXt block.

SE-Net

The SE-Net architecture introduces the idea of modeling dependencies between different feature maps of one convolutional layer (Hu et al. 2017). This adds connection between different feature maps within a layer using trainable weights for the connection. The architectural network blocks, which were introduced to achieve this effect, are called *Squeeze-and-Excitation blocks* (S&E blocks). The squeeze blocks generates global information about each feature maps of a layer by calculating the average activations for all feature maps for that layer. The excitation function is a small neural network itself that learns weights for a fully connected layer that takes the squeeze block outputs as inputs and thereby models the connection between different features maps within a layer. This results are a modeled dependency between layer activations that was until now disregarded. Using those SE-blocks, the accuracy can be improved under the ceteris paribus assumption (Hu et al. 2017).

Model Comparison

Classification Results

The top-5 misclassification rate decreases throughout the years from 2012 to 2017 (cf. Table 1). When looking at the magnitude of decreasing misclassification, the results in the years from 2015 on only increase marginally and are only relevant to the benchmark itself.

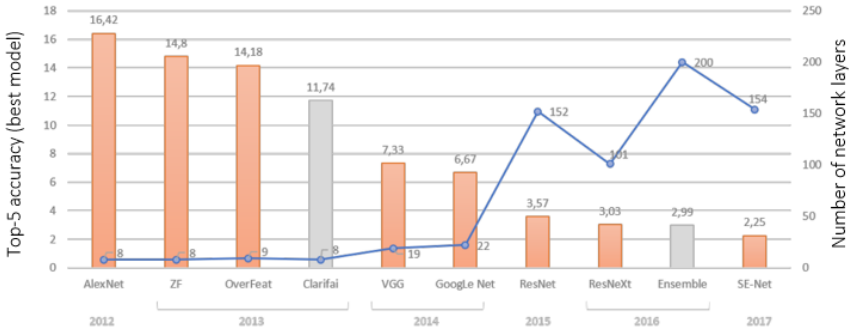


Figure 7. Model Misclassification and Number of Layers

Using human benchmarks on the benchmark dataset, it was found that the human top-5 accuracy in object recognition lies at approximately 5% (Russakovsky et al. 2015). Therefore, the architectures using residual blocks and SE-blocks are able to outperform human object recognition.

In addition, it was found that CNN misclassification could be traced back to certain patterns: 21% of misclassified images could be traced back to small objects, 13% to color modifications through filters and 0.03 % to mislabeled images which has some implications on the training process in return (Russakovsky et al. 2015). Figure 7 depicts the evolution of misclassification rate (left axis, bars) vs. the number of layers (right axis, blue line). The models with grey bars are depicted for completeness. However, they were not considered in our study since they are ensembles of existing models and therefore do not contribute new architectural features to evaluate.

Model Size

In terms of model size, Figure 8 depicts the top-1 accuracy performance per one million parameters to compare models based on size with regard to performance. The models are ordered by information density, which is given by the ratio of accuracy and number of parameters. The smallest information densities are provided by the VGGNet and OverFeat architectures, which is caused by their huge parameter count. To improve comparison, we added the MobileNet architecture, which was introduced for mobile computing and therefore is limited to lightweight computing resources in its architecture (Howard et al. 2017). It was

specifically generated to model the trade-off between accuracy and inference time by organizing the convolutional layers in separate blocks to scale the model.

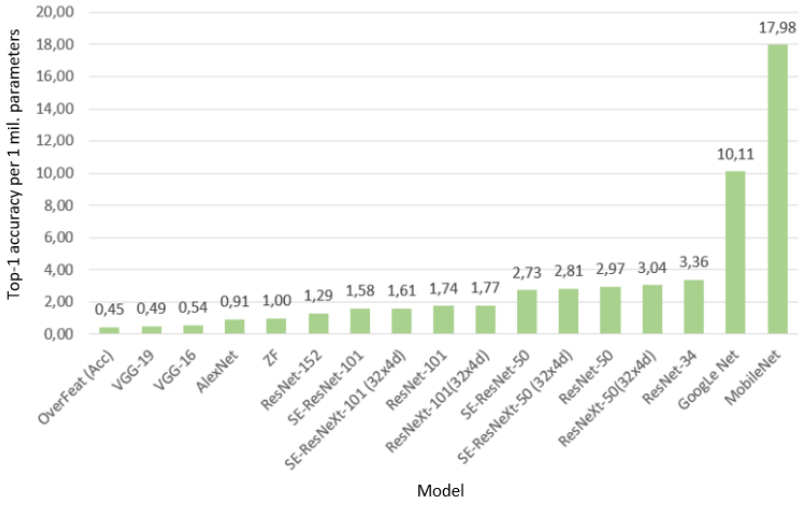


Figure 8. Ratio of Accuracy Performance to Number of Layers (Information Density)

The MobileNet architecture achieves trade-off values in terms of accuracy for the benchmark dataset and was therefore not among the winners and hence excluded during our primary model collection. The original MobileNet consists of 28 layers, 4.2 million trainable parameters and achieves a top-1 accuracy of 70.60 % with an inference time of 569 million flops. Comparing the GoogLeNet and the VGGNet-19, we can see a major difference between 10 % and 0.50 % information density. However, the VGGNet-19 achieved better accuracy values.

Model	Model Memory	Feature Memory	Model	Model Memory	Feature Memory
AlexNet	229	**4**	ResNet-152	230	80
ZF-Net	238	8	ResNeXt-50 (32x4d)	98	47
OverFeat Acc.	549	9	ResNeXt-101 (32x4d)	170	73
VGGNet-16	496	52	ResNeXt-101 (64x4d)	319	141
VGGNet-19	549	57	SE-ResNet-50	107	36
GoogLeNet	26	15	SE-ResNet-101	188	56
ResNet-34	83	15	SE-ResNeXt-50 (32x4d)	107	47
ResNet-50	98	36	SE-ResNeXt-101 (32x4d)	188	73
ResNet-101	170	56	MobileNet	**16**	38

Table 2. Model Memory Requirements Comparison in Megabyte

To compare the memory needs, we calculate *i) feature memory requirements* using the number of feature values and *ii) model memory requirements* using the parameter counts. For the MobileNet and the ResNeXt-101 (64×4d) architecture, we do not have comprehensive values from literature and therefore use the reference values given by (Albanie 2017). Table 2 shows the memory requirements of the different models in megabyte. Here, we can identify the trend that the models with the highest feature memory requirements produce the best top-1-accuracy values (with the exception of the VGGNet models). Furthermore, the OverFeat and the VGGNet model require by far the most model memory, which can explain their low information density in Figure 8.

Inference Time

The inference time comparison of the models is depicted in Figure 9, where the models are ordered by top-1 model accuracy from left to right.

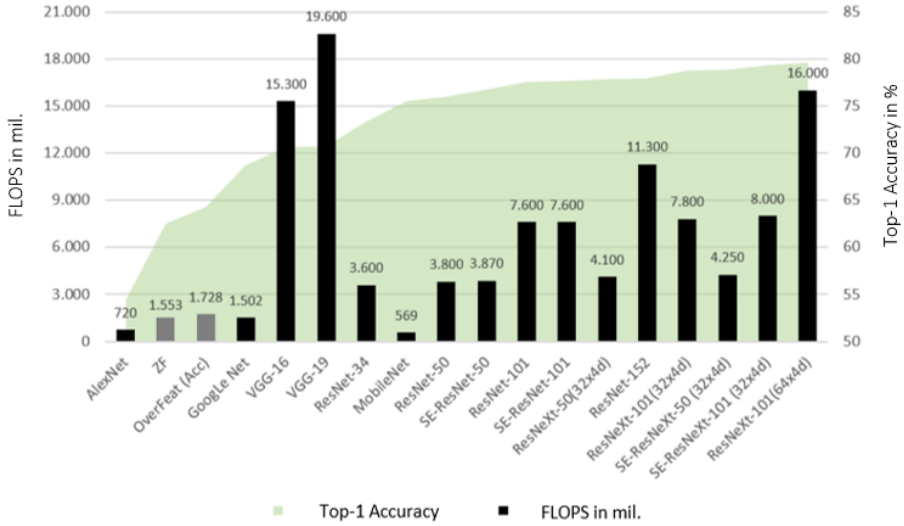


Figure 9. Inference Time Comparison of Models in Million FLOPS

The inference time measures the time (in FLOPS) that is needed to perform a forward run to classify new data. As such, it gives a magnitude on the time it takes for the model to solve the given problem within a business context. It can also be used as a constraint on model architecture.

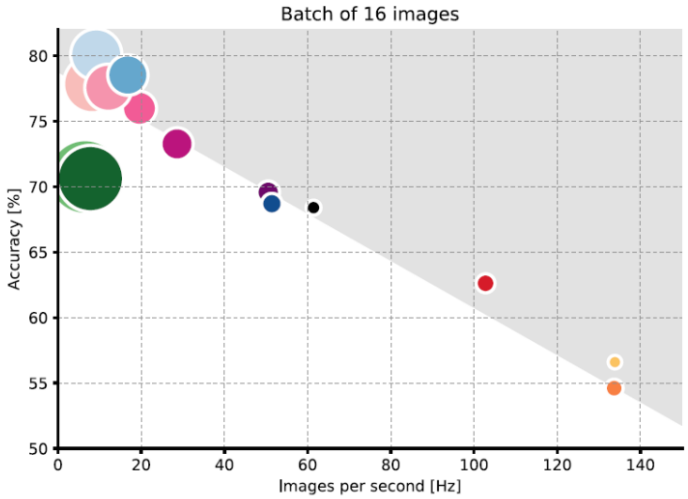


Figure 10. Capacity vs. Accuracy (Canziani et al. 2016)

The relative relationship among the models, considering the time criteria, is similar to the results we found when comparing the memory requirements. It can be shown in general that more convolution operations lead to increased inference times. Especially the lean MobileNet architecture shows that higher accuracies can be reached with lower inference times by modifying the architecture (Howard et al. 2017). If a business context has constraints on runtime, a leaner version of an architecture involving fewer parameters can be used. In general, we observe a trade-off between model accuracy and inference time. This result is also in accordance with previous conducted research on inference time by Canziani et al. (2016). In the previous research, the capacity of a model is measured by frames per second while measuring the top-1 accuracy. This relationship is depicted in Figure 10, where the size of the areas reflects the FLOP count. A linear dependency between inference time and accuracy can thereby be confirmed, so that the line constitutes a

boundary on this trade-off. Therefore, models that lie above the line in the grey area depict models with a more favorable trade-off combination (Canziani et al. 2016).

Discussion

Evolution of CNN Architectures

When looking at the evolution of CNNs from 2012 (i.e., when AlexNet was introduced as a remarkable type of network architecture to the image recognition benchmark data, which we considered for our investigation), the foundational architecture parts basically remained unchanged over the last few years' developments (Krizhevsky et al. 2012). As such, the CNN architecture always consisted of a feature extractor part with convolutional and pooling layers and a classifier part following the fully connected layers (LeCun et al. 2015). In subsequent years, a modular toolbox followed that can be stacked and connected in different ways.

Our model comparison reveals that the best models applied to the ILSVRC benchmark data were mostly built upon existing models from previous years. We can therefore describe the development of CNNs as a gradual evolutionary process in which enhancements focus on very particular aspects of the network architecture instead of introducing a revolutionary and novel architectural structure. Those aspects include, for example, the optimization and introduction of new hyperparameters or the introduction of a new way to connect and stack layers and neurons within a CNN.

The first model improvements following the AlexNet architecture resulted from layer diagnostics using deconvolution and finding that a smaller filter size with a smaller stride in the first layers can improve classification performance (Zeiler and Fergus 2013). Following this deconvolution approach dealing with filter sizes, the next step incorporated experiments with the network depth, not only concerning single hyperparameters but rather the whole network architecture. The results of the experiments showed that deeper networks can achieve improved classification results, which were used in the OverFeat and VGGNet architectures (Sermanet et al. 2013; Simonyan et al. 2013). Additional improvements were made by stacking an increasing number of convolutional layers and thereby generating even deeper CNN architectures.

The inception module, as introduced in GoogLeNet (Szegedy et al. 2015), improved the network by stacking and connecting the convolutional layers in a different way. Instead of stacking layers, the convolution layers are stacked within one capsuled layer. To make model training feasible, the depth of the resulting feature maps was reduced by enforcing small 1×1 filters within the inception module.

It was then found by He et al. (2015) that deeper architectures suffer from problems in the more advanced layers, where network weights tend to converge towards zero in those layers and therefore have small influence on the classification. This problem was solved by re-organizing the layers into residual blocks that allow shortcut connections to bypass the degradation and vanishing gradient problems in the ResNet (He et al. 2015). This idea was further improved by, again, stacking and reorganizing the layers in a parallel fashion within a residual block, instead of sequentially stacking them (Xie et al. 2016). The main goal of this ResNeXt architecture was to improve training and inference time while maintaining network depth. An additional improvement of the ResNet architecture was made by introducing the SE-Net that allows for connections between feature maps of one layer in a Squeeze-and-Excitation block. This block can be used in every network architecture and, ceteris paribus, improves classification results (Hu et al. 2017). Additional developments in this research field of designing CNN architectures, which do not directly influence benchmark results, are given improvements in pooling functions using spectral or stochastic pooling and improved activation functions (Gu et al. 2018). Furthermore improvements are made regarding the training process and finding new use cases for the application of CNN, such as mobile applications (Howard et al. 2017).

Guidelines for Design and Architecture Choice of CNN

The general idea of designing a CNN is to solve a particular classification task and achieve high accuracy values with a given architecture. However, because of the complex nature of DLN architectures, finding the optimal architecture for a given problem is based on a guided stochastic scientific process. Our findings suggest that we can improve the guided part of this process by learning some design principles from the evolutional comparison that we conducted in our research. We summarize them as design guidelines in

Table 3. To structure the findings and practical design implications we use the guideline constraints that are proposed by Cao (2015): *capacity of learning* which refers to guidelines for designing convolutional networks from the (A) layer perspective and *necessity of learning* regarding the (B) network /connection perspective (see Section "Architectural Principles of CNNs" for details). Since the evolution exposed some general problems with deep architectures in terms of results quality and model inference time, we formulate two additional constraints on (A) and (B): *effectiveness of learning* and *efficiency of learning*.

Design constraint		Design guideline	Literature
Scope of Learning	*Capacity of Learning*	Use detailed filter kernel including smaller strides **BUT** Set lower bound in order to capture complex image parts	(Zeiler and Fergus 2013) (Cao 2015)
	Necessity of Learning	Set upper bound on model depth so that the topmost convolutional layer is no larger than the input	(Cao 2015; Simonyan and Zisserman 2014)
	Effectiveness of Learning	Allow shortcut connections by using residual blocks **AND** Allow inter-feature map connections in a single layer using S&E blocks	(He et al. 2015) (Hu et al. 2017)
	Efficiency of Learning	Allowing block structure by using inception modules	(Howard et al. 2017; Szegedy et al. 2015)

Table 3. Design constraints and design principles for DLN

Capacity of Learning: The *capacity of learning* describes the quality of a CNN to capture complex spatial features. This depends on filter size parameters and stride. Choosing a small filter with large down-sampling by stride values larger than one can lead to the loss of detailed representation of the input image regions. However, using increasingly detailed filter kernels diminishes the capability of representing complex spatial features that can be typical for a given object (e.g. position of tires relative to the rest of the car). Therefore a trade-off between detail and capturing complexity has to be made and the size of the filter kernel and stride can be determined by setting a minimum complexity value that emerges as a ratio between the filter size (including stride as a multiplication factor) and the size of the receptive field that reflects the portion of the input image the feature map values looked at. Cao (2015) suggest a lower bound ratio value of 1/6.

Necessity of Learning: The *necessity of learning* refers to the stopping condition regarding the depth of the network. While it is suggested that deeper networks yield better results in general, the improvements in additional layers beyond a certain points are marginal and can generate the problem of overfitting (He et al. 2015; Szegedy et al. 2015). It is therefore suggested to limit the depth of a network based on the increasing size of the receptive field, since new, more complex patterns fail to emerge once the network has seen the whole image. We implement and follow the suggestion by Cao (2015) to limit the model depth and therefore the size of the topmost convolutional layer to be no larger than the actual image size. It is important to note that the constraint on the depth of the network is only determined by the receptive field of the topmost layer. That only concerns the rule of not adding an additional layer after full receptive field size is reached. However, it does not state how many layers are required to reach that.

Effectiveness of Learning: Next, the *effectiveness of learning* constraint reflects on classification quality. We found that introducing residual blocks and bottleneck residual blocks into a CNN architecture with an increasing number of convolutional layers improves performance by eliminating or diminishing both, the degradation and vanishing gradient problem (He et al. 2015). Additionally, S&E blocks can be used to further increase performance (Hu et al. 2017).

Efficiency of Learning: Beginning with the architecture of GoogLeNet, researchers started to include constraints on inference time a network architecture. Therefore we introduce the additional design constraint of *efficiency of learning*. As stated in Szegedy et al. (2015) and Howard et al. (2017) using inference time constraints lead to the introduction of inception modules that organize convolutional layers in a block structure which leads to a substantial decrease in inference time.

Scope of Learning: When applying CNNs to real world problems, one can use the introduced architectures since they perform well on a given benchmark dataset. However, the input data may need some adaption of the model in terms of layer structure and depth. In general the above mentioned guidelines are only a small part in the stochastic design process of finding the optimal DLN architecture for a specific problem. It is therefore important to introduce an overarching constraint that is determined by the problem at hand: the *scope of learning*. This constraint reflects on the dependency of model architecture and therefore the execution of our guidelines on the practical problem at hand. Nearly all of the restrictions in reflected by upper and lower bounds on network architecture depend on the model input (e.g., the maximum size of the receptive field). It is therefore important to stretch out that one always has to adjust and scale the architecture with the input data size and type, that is the VGG-19 architecture net might perform very well on the ImageNet data set but will fail at French language translation (type) and will therefore not be effective, likewise, the GoogLeNet might perform well at the ImageNet classification task but would be an "overkill" architecture for the handwriting recognition task (MNIST) and therefore not be efficient.

In general, without any constraints on efficiency of learning, it can be stated that regarding the effectiveness of learning, ensemble models usually outperform single models. Imposing constraints on the efficiency of learning, critically influences the choice of architectural parameters.

Exemplary application of guidelines: An example are real-time or near real-time application of outcome failure prediction in a manufacturing chain. While the prediction of imminent failure is the primary task, it has to be done with regard to time constraints that are imposed by the business and manufacturing processes. As a result, the network inference time is restricted to be very small. In conclusion, a user of the failure detection system would prefer a network that detects a subset of errors with inferior performance, rather than a model that exhibits better performance but cannot output the failure detection decision in the required time.

Summarizing, the design of the CNN is the prime driver of inference time and accuracy. However, with the evolution of DLN architectures, this trade-off can be skewed. An example for such a skewed trade-off is given in Iandola et al. (2016), where accuracy values comparable to the AlexNet evaluation values could be achieved with only using 2 % of the parameters.

Conclusion & Outlook

In this paper, we compared several architectures that were applied to a benchmark dataset to ensure comparability. We defined three main criteria: *(i)* model size, *(ii)* classification results, and *(iii)* time of inference and subsequently evaluated different CNN architectures based on those criteria. We then gave a discussion that both summarized the evolutional development of those models and provided some initial guidelines on how to design CNN architectures given the trade-off between model accuracy and inference time. Limitations of our work include the considered model base, which did not include ensemble models and was chosen based on only one benchmark dataset for scientific comprehensibility. Hence, our approach described lacks completeness. Nevertheless, the models were chosen based on their performance and not on their architectural complexity. Therefore, we later also added a model that solely focuses on low inference time based on the idea to apply it to mobile applications. In addition, for the bigger picture, we have to acknowledge that there are many other factors that influence prediction results that we did not focus on like training data set and validation data set size, since they are factors that are independent from the architecture. Despite these limitations, the models allowed us to derive guidelines for CNN design and architecture choices as a methodological contribution. Future research should update the model base and also include the principle of transfer learning, answering the question of how many data we need to adjust general models for a specific problem without retraining them from scratch.

References

Albanie. 2017. "Memory Consumption and FLOP Count Estimates for Convnets: Albanie/Convnet-Burden," , August. (https://github.com/albanie/convnet-burden, accessed October 12, 2018).

Canziani, A., Paszke, A., and Culurciello, E. 2016. "An Analysis of Deep Neural Network Models for Practical Applications," *ArXiv:1605.07678 [Cs]*. (http://arxiv.org/abs/1605.07678).

Cao, X. 2015. "A Practical Theory for Designing Very Deep Convolutional Neural Networks," *Technical Report*.

Cheng, J., Wang, P., Li, G., Hu, Q., and Lu, H. 2018. "Recent Advances in Efficient Computation of Deep Convolutional Neural Networks," *Frontiers of Information Technology & Electronic Engineering* (19:1), pp. 64–77. (https://doi.org/10.1631/FITEE.1700789).

Glorot, X., and Bengio, Y. 2010. *Understanding the Difficulty of Training Deep Feedforward Neural Networks*.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. 2018. "Recent Advances in Convolutional Neural Networks," *Pattern Recognition* (77), pp. 354–377. (https://doi.org/10.1016/j.patcog.2017.10.013).

He, K., Zhang, X., Ren, S., and Sun, J. 2015. "Deep Residual Learning for Image Recognition," *ArXiv:1512.03385 [Cs]*. (http://arxiv.org/abs/1512.03385).

Heinrich, K., and Fleissner, V. 2018. "Deep Intelligent Systems for Time Series Prediction: Champion or Lame Duck? - Evidence from Crude Oil Price Prediction," *AMCIS 2018 Proceedings*. (https://aisel.aisnet.org/amcis2018/Semantics/Presentations/9).

Heinrich, K., Roth, A., and Zschech, P. 2019. "Everything Counts: A Taxonomy of Deep Learning Approaches for Object Counting," in *Proceedings of the Twenty-Seventh European Conference on Information Systems*, Uppsala, Sweden.

Heinrich, K., Zschech, P., Möller, B., Breithaupt, L., and Maresch, J. 2019. "Objekterkennung im Weinanbau – Eine Fallstudie zur Unterstützung von Winzertätigkeiten mithilfe von Deep Learning," *HMD Praxis der Wirtschaftsinformatik*. (https://doi.org/10.1365/s40702-019-00514-9).

Heinrich, K., Zschech, P., Skouti, T., Griebenow, J., and Riechert, S. 2019. "Demystifying the Black Box: A Classification Scheme for Interpretation and Visualization of Deep Intelligent Systems," *AMCIS 2019 Proceedings*. (https://aisel.aisnet.org/amcis2019/ai_semantic_for_intelligent_info_systems/ai_semantic_for_intelligent_info_systems/8).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. 2017. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *ArXiv:1704.04861 [Cs]*. (http://arxiv.org/abs/1704.04861).

Hu, J., Shen, L., and Sun, G. 2017. "Squeeze-and-Excitation Networks," *ArXiv:1709.01507 [Cs]*. (http://arxiv.org/abs/1709.01507).

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. 2016. "SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and \textless0.5MB Model Size," *ArXiv:1602.07360 [Cs]*. (http://arxiv.org/abs/1602.07360).

Krizhevsky, A., Sutskever, I., and Hinton, G. E. 2012. "Imagenet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, pp. 1097–1105. (http://papers.nips.cc/paper/4824-imagenet-classification-w).

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. 2007. "An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation," in *Proceedings of the 24th International Conference on Machine Learning*, ACM, pp. 473–480.

LeCun, Y. 1989. "Generalization and Network Design Strategies," in *Connectionism in Perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels (eds.), Elsevier.

LeCun, Y., Bengio, Y., and Hinton, G. 2015. "Deep Learning," *Nature* (521:7553), pp. 436–444. (https://doi.org/10.1038/nature14539).

Mishkin, D., Sergievskiy, N., and Matas, J. 2017. "Systematic Evaluation of CNN Advances on the ImageNet," *Computer Vision and Image Understanding* (161), pp. 11–19. (https://doi.org/10.1016/j.cviu.2017.05.007).

Mody, M., Kumar, D., Swami, P., Mathew, M., and Nagori, S. 2018. "Low Cost and Power CNN/Deep Learning Solution for Automated Driving," in *2018 19th International Symposium on Quality Electronic Design (ISQED)*, , March, pp. 432–436. (https://doi.org/10.1109/ISQED.2018.8357325).

Pianpanit, T., Lolak, S., Sawangjai, P., Ditthapron, A., Marukatat, S., Chuangsuwanich, E., and Wilaiprasitporn, T. 2019. "A Comparative Study for Interpreting Deep Learning Prediction of the Parkinson's Disease Diagnosis from SPECT Imaging," *ArXiv Preprint ArXiv:1908.11199*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. 2015. "ImageNet Large Scale Visual Recognition

Challenge," *International Journal of Computer Vision* (115:3), pp. 211–252. (https://doi.org/10.1007/s11263-015-0816-y).

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. 2013. "OverFeat: Integrated Recognition, Localization and Detection Using Convolutional Networks," *ArXiv:1312.6229 [Cs]*. (http://arxiv.org/abs/1312.6229).

Simonyan, K., Vedaldi, A., and Zisserman, A. 2013. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," *ArXiv:1312.6034 [Cs]*. (http://arxiv.org/abs/1312.6034).

Simonyan, K., and Zisserman, A. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition," *ArXiv:1409.1556 [Cs]*. (http://arxiv.org/abs/1409.1556).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. 2015. "Going Deeper with Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. 2016. "Aggregated Residual Transformations for Deep Neural Networks," *ArXiv:1611.05431 [Cs]*. (http://arxiv.org/abs/1611.05431).

Zeiler, M. D., and Fergus, R. 2013. "Visualizing and Understanding Convolutional Networks," *ArXiv:1311.2901 [Cs]*. (http://arxiv.org/abs/1311.2901).