Master Thesis

Evaluation of the Transferability in Deep Networks

written by

Florian Bemmerl

Matrikel: 0405722

First Examiner:	Prof. Dr. Klaus Obermayer, TU Berlin
Second Examiner:	Prof. Dr. Manfred Opper, TU Berlin
Supervisor:	Thomas Goerttler, TU Berlin

Technical University Berlin, Electrical Engineering and Computer Science

Institute of Software Engineering and Theoretical Computer Science
Neural Information Processing Group

Berlin, January 2021

Contents

Abstract 1

1 Introduction 3
- 1.1 Research Question . . . 5
- 1.2 Related Works . . . 5
- 1.3 Thesis Structure . . . 7

2 Methodology 9
- 2.1 Architectures and Datasets . . . 9
  - 2.1.1 Convolutional Neural Networks . . . 9
  - 2.1.2 VGG Networks . . . 10
  - 2.1.3 Datasets . . . 12
- 2.2 Transferability Metrics . . . 23
  - 2.2.1 Intrinsic Dimension (ID) . . . 23
  - 2.2.2 Sum of Squares (SS) . . . 25
  - 2.2.3 Representational Similarity Analysis (RSA) . . . 26
- 2.3 Experimental Setup . . . 28
  - 2.3.1 Pre-training . . . 28
  - 2.3.2 Extract Activations . . . 31
  - 2.3.3 Fine-tuning . . . 32

3 Results 35
- 3.1 Results for CNN architecture . . . 35
  - 3.1.1 Pre-training . . . 35
  - 3.1.2 Evaluate metrics . . . 36
  - 3.1.3 Transfer learning outcomes . . . 39
- 3.2 Results for VGG architecture . . . 43
  - 3.2.1 Pre-training . . . 43
  - 3.2.2 Evaluate metrics . . . 43
    - 3.2.2.1 Custom3D . . . 44
    - 3.2.2.2 Malaria . . . 46
    - 3.2.2.3 Oxford Pet . . . 48
  - 3.2.3 Transfer Learning outcomes . . . 50
- 3.3 Evaluate Transferability Metrics . . . 54

4 Conclusions **65**
4.1 Review of Research Question 65
4.2 Contributions 66
4.3 Future Work 67

A Appendix **69**
A.1 Developed program code 69
A.1.1 Environments, Frameworks, and Packages 69
A.1.2 Program Code 69
A.2 Additional Figures 72

List of Figures **76**

List of Tables **77**

Bibliography **79**

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle sinngemäß und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

Berlin, den 20. Januar 2021

Unterschrift

Abstract

This thesis investigates the workings of transfer learning in deep neural networks for supervised visual classification tasks and contributes to the open question of transferability estimation. Specifically, the goal is to propose new transferability metrics, derivable only from the source model and a subset of the target dataset, and study their capabilities to be used for predicting transfer learning performance, i.e. the accuracy archived on a novel target task when using pre-trained representations of a source task as initialization. We find strong correlations for all three investigated metrics, namely intrinsic dimension, sum of squares and representational similarity analysis. Therefore adding empirical weight to the assumption that the quality of learned representations as initialization for a different task can be quantified before transfer and relates to their representational similarity.

Kurzfassung

Diese Arbeit untersucht die Funktionsweise des Transferlernens in neuronalen Netzen für visuelle Klassifikation und trägt zur Untersuchung der offenen Frage der Übertragbarkeitsschätzung bei. Konkret geht es darum, neue Übertragbarkeitsmetriken vorzuschlagen, die nur aus dem ursprünglichen Modell und einer Teilmenge des Zieldatensatzes abgeleitet werden können, sowie zu untersuchen ob sie für die Vorhersage der Transferlernleistung verwendet werden können, d. h. die Genauigkeit, die für eine neuartige Zielaufgabe erzielt wird, wenn vorher trainierte Repräsentationen einer Ursprungsaufgabe als Initialisierung verwendet werden. Wir erhalten starke Korrelationen für alle drei untersuchten Metriken, namentlich intrinsische Dimension, Quadratsumme und Repräsentationsähnlichkeitsanalyse. Damit wird die Annahme empirisch untermauert, dass die Qualität von gelernten Repräsentationen als Initialisierung für eine andere Aufgabe vor dem Transfer quantifiziert werden kann und mit ihrer repräsentationalen Ähnlichkeit zusammenhängt.

1 Introduction

Within the multitude of artificial intelligence, methods deep learning has undoubtedly attracted the greatest attention in recent time. In a wide range of research, new state-of-the-art results are published repeatedly and applications in numerous real-world problems show success, even exceeding human-level performance in many cases.

In contrast to traditional machine learning techniques, for which features had to be designed manually, deep learning algorithms try to learn high-level features from mass data. As a representation learning algorithm requiring large-scale datasets to understand latent patterns in data, deep learning's biggest problem is data dependency. Massive training datasets and large-scale models are necessary for sufficient learning performance on complex data.

An almost linear relationship between model size and amount of required data can be found and is commonly explained by the concept "that for a particular problem, the expressive space of the model must be large enough to discover the patterns under the data" [Tan et al., 2018].

The foundations of convolutional neural networks on visual tasks have been set for decades, e.g. [LeCun et al., 1989]. But only once the research and technology had progressed far enough, the ImageNet Large Scale Visual Recognition Challenge [Russakovsky et al., 2015], which has attracted many computer vision research groups from around the world both in academia and industry since 2009, was won by a deep learning approach in 2012 by the research group lead by Hinton [Krizhevsky et al., 2012]. As they state themselves: only due to [Deng et al., 2009] creating the ImageNet dataset, "the essential missing ingredient was supplied by FeiFei et al. who put a huge effort into producing a labeled dataset that was finally large enough to show what neural networks could really do" [Krizhevsky et al., 2012].

Ushering in an era of dominance for deep learning on supervised visual classification tasks and beyond. But the collection of labeled training data, annotated for the task to learn, is an inevitable challenge, in that it's complex and expensive. Making it extremely hard to collect a large-scale dataset sufficient for deep learning. Depending on the domain it's also not just the price of paying for human labeling but e.g. in medical applications, the gathering of more data could require longer, complex clinical trials or even more patients in pain.

Transfer Learning helps to mitigate the problem of data dependency by first training a model on a task with sufficient training data, before transferring it to a different target

task. Therefore expanding the range of possible training data and in that sense relaxing the requirement of independent and identically distributed (i.i.d.) training and test data, allowing for strong learning performance on the target task with limited data samples. The assumption that deep learning models are capable of learning invariant representations that are transferable across tasks, therefore allows us to significantly reduce the demand of training data and training time in the target domain by using transfer learning and not starting training a model from scratch.

As mentioned above, transfer learning is an important technique to mitigate the manual labeling effort in machine learning and computer vision, like in [Pan et al., 2010], [Wang and Schneider, 2014] and [Long et al., 2013]. By transferring knowledge between the domains of the training / source task and the target task, bridging the domain discrepancy in the probability distributions of different domains is the main challenge of effective transfer.

Deep learning methods are capable of not just learning shallow features but disentangle explanatory factors of variance in the data samples by learning nonlinear representations [Bengio et al., 2013]. These deep representations describe underlying invariant data properties and are transferable from one task to similar novel tasks [Yosinski et al., 2014].

The general assumption transfer learning is based on is neuroscience-inspired, in the way that a neural network is expected to behave similar to the processing mechanism of a human brain, having a layered, iterative, and continuous abstraction process. This leads to the common perception of the early layers of a network being a general feature extractor with versatile features, that are thus preferred for transfer, and later specialized layers combining the features to derive the output decision, seen as the classifier.

In particular, we use network-based deep transfer learning in the following, one of the major categories of transfer learning based on the classification by [Tan et al., 2018]. It refers to the partial reuse of a network pre-trained in "the source domain, including its network structure and connection parameters, [and transferring] it to be a part of a deep neural network which is used in the target domain." [Tan et al., 2018]. Furthermore, the process can be categorized as inductive transfer learning, since labeled data samples are necessary for the source and target data to pre-train and fine-tune in a supervised manner in both domains.

Transfer learning is an interwoven aspect of machine learning with a multitude of domains touching upon topics from meta-learning, to joint distribution adoption or representational learning. All of the aspects are still very actively pursued in academic research and used daily in many DNN applications to improve performance or reduce computational and data requirements through transfer. And while there are more and more pre-trained models available, one of the major open questions of transfer learning that provoked our thought and gives reason to this thesis is, how to select the model initialization that archives the highest performance on a new target task out of several pre-trained ones?

1.1 Research Question

In this work, we set out to gain insight into the workings of deep transfer learning for supervised visual classification tasks and specifically strive to introduce a transferability metric capable of predicting transfer performance without the need of actually conducting transfer and fine-tuning of weights in the target domain.

In order to archive this, three candidate metrics are introduced as well as a setup with multiple architectures and datasets to test and evaluate the predictive quality of the metrics.

The metrics are calculated on the activations of models trained on source tasks when fed with samples of the target task, but without any form of transfer. But to be able to assess the metrics, the 'ground-truth' of knowing which pre-trained model actually archives the highest transfer performance on the target task is necessary, therefore conducted as well but separately. Correlating our metrics to these results then yields the assessment of transferability.

Our proposed metrics and methodology builds and expands upon previous works on model selection for transfer learning. And our work for one is inspired by [Yosinski et al., 2014], which comprehensively explores the feature transferability of DNNs, as well as the very recent works of [Nguyen et al., 2020] and [Dwivedi et al., 2020], that similarly set out to introduce measures and frameworks for model selection in deep transfer learning.

1.2 Related Works

Using transfer learning to better deal with a challenging target task, by selecting a pre-trained model to use as initialization is a common technique. Nevertheless, the question that imminently arises and that we pose for this thesis as well, is not thoroughly answered yet: how does one choose the ideal model initialization resulting in the highest accuracy performance when learning a new task?

With their paper 'Taskonomy: Disentangling Task Transfer Learning' [Zamir et al., 2018] provided a pioneering work attempting to disentangle the relationship between visual tasks computationally and provided the base for much of the research conducted since regarding transferability. Laying out a dictionary of 26 visual tasks (2D, 2.5D, 3D and semantic) and proposing a fully computational approach based on training all the task models and all feasible transfers between them fully supervised. Obtaining an affinity matrix of transferability, upon which a task-transferability graph is derived, leading to remarkable results that suffer from extensive training costs though. Especially since adding a new task requires a combinatorial effort, of the transferability having to be trained between the new task and all others in the graph.

Since 2018, papers following the example of Taskonomy have tackled transferability estimation, in the sense of addressing the problem of predicting the performance of transfer learning between two classification tasks, without actually executing it and therefore assisting the selection of initialization. Most noticeably, [Dwivedi and Roig, 2019] using representational similarity analysis and [Song et al., 2019] using attribution maps, leveraged the results for transfer learning performance of the Taskonomy datasets as ground-truth to develop and benchmark their methods for model selection. Both outperformed the Taskonomy results.

Since we are also going to use Representational Similarity Analysis (RSA) in our work the results of [Dwivedi and Roig, 2019] are a major inspiration. And we build upon their argument of comparing DNN representations using RSA, to transfer weights from a model with representations similar to the new task's representations, will lead to higher transfer performance than for models with less similar representations.

Further research based on task embedding on meta-learning was conducted, e.g. by [Achille et al., 2019], trying to derive distances between tasks, that also answers to the transferability estimation problem. And [Tran et al., 2019] developed another approach with the negative conditional entropy measure between only the source and target label sets. Providing a solution agnostic technique that does not require trained models and instead is based on information theory. Another related work worth mentioning is the thorough theoretical analysis towards understanding the transferability of deep representations by [Liu et al., 2019].

The two most recent works [Nguyen et al., 2020] and [Dwivedi et al., 2020] are also the most related works to our approach, showing the current relevance and active research this thesis is in line with.

The Log Expected Empirical Prediction (LEEP) for transferability estimation of deep networks proposed by [Nguyen et al., 2020] it a simple classifier that makes predictions based on the expected empirical conditional distribution between source and target labels. Like our approach, its scores can be obtained from just a source model and a target data set by a single forward pass of the target data through the model. Therefore providing a much simpler process than previous methods as well as easier interpretation. LEEP can be applied to plenty of deep networks in general settings and so can our metrics, e.g. they do not need to follow strict requirements like in [Tran et al., 2019], instead allow arbitrarily different input distributions.

[Dwivedi et al., 2020] introduced a generic framework for initialization selection in task transfer learning with their Duality Diagram Similarity (DDS) metric. Following their work with RSA on Taskonomy mentioned above [Dwivedi and Roig, 2019], it builds upon the same assumption that higher similarity between representations, leads to higher transfer learning performance, that we also pursue. They actually show that their previous measure, RSA, can be posed as a particular case in the general DDS formulation. In DDS a matrix of features is obtained from feedforwarding target images through a DNN again and extended

with a matrix quantifying the dependencies between the individual feature dimensions and another one that assigns weights to the observations. Resulting in a representation of a deep network for a set of inputs being able to be expressed by its duality diagram. Then comparing the duality diagrams of two models a similarity score is obtained and finally the ranking of the scores is correlated to the ranking of transfer learning performance using Spearman's rank correlation. This allows comparing the source tasks ranking based on the metric with the ranking based on transfer learning performance, the same way we also conduct our evaluation.

As [Dwivedi et al., 2020] states in regards to all the previous works on transferability estimation: "due to lack of a standard benchmark with standard evaluation metrics, comparing and building upon these methods is not trivial". And so we going to build upon the aforementioned works and try to go beyond by introducing new metric candidates.

1.3 Thesis Structure

As the context of the work and especially the background of transfer learning on deep neural networks has been introduced at the beginning of this chapter. We set out to investigate the research question of Section 1.1 in the following chapters, building upon many of the ideas mentioned in the related work literature discussion of Section 1.2.

In Chapter 2 the methodology for setting up and conducting experiments to explore transferability are explained. Including descriptions of the network architectures used (CNN and VGG) in Section 2.1, as well as the source and target task datasets (in Section 2.1.3 specifically). In Section 2.2 the three candidates for a transferability metric are introduced and their potential for the insight into the workings of DNNs explained. Finishing up Chapter 2 is the detailed outline of the process to conduct the empirical experiments, divided into the three main steps of pre-training, extracting the activations from the models, and fine-tuning on the target task.

These lead us into Chapter 3, where the results obtained from the experiments are stated, visualized and analyzed regarding the workings of transfer learning. It is split by the architecture used into Section 3.1 and Section 3.2, followed by the main evaluation and assessment of the findings by reporting on the correlations between metrics and transfer performance in Section 3.3.

Chapter 4 concludes with a review of the research objective and an assessment of the contributions made to the cause of finding a transferability metric and the results archived. Closing with an outlook on future work remaining after this thesis.

The Appendix contains additional graphs and visualizations, as well as parts of the program code and technical details. And a list of all figures and tables as well, as the bibliography with all references can be found at the very end.

2 Methodology

This chapter sets out to introduce and thoroughly explain all the components of the methodology applied in the following of this thesis to obtain and evaluate results that will contribute to investigating the research objective.

First, in Section 2.1 the architectures of the neural network models used are explained, followed by an introduction of all the datasets used for the source and target tasks.

Then the three transferability metric candidates: intrinsic dimension, sum of squares and representational similarity analysis are introduced and defined in Section 2.2.

Finally, the experimental setup of how each of the previous elements fit into place for the actual training of models, extraction of activations and calculation of the metrics, as well as the transferring and fine-tuning for each of the target tasks is explained and the parameters and settings necessary for replicating the results are given in Section 2.3.

2.1 Architectures and Datasets

Two different model architectures are used for conducting the experimental analysis of transfer performance. Either a small convolutional neural network (Section 2.1.1) or a very deep VGG model (Section 2.1.2) are used, depending on the complexity of the multitude of datasets selected for comparison (Section 2.1.3). To fix some hyperparameters and primarily to ensure comparability, the architectures used are fixed for all experiments, except for the output layer, since different tasks require varying output dimensionality.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks are inspired by the visual cortex and employ convolutions to extract visual features automatically from raw pixel data through sliding filter windows scanning the image, unlike the classical image recognition where image features had to be defined by hand. Research started in the 1980s, most noticeably [LeCun et al., 1989] and with

breakthroughs in data mining and GPU development, it placed itself as the predominant algorithm for visual tasks since [Krizhevsky et al., 2012].

There are many configurations of CNN architectures, ones with only one convolutional layer or very deep ones like VGG (see next Section 2.1.2) with dozens. Going forward we define a small convolutional network with two convolutional layers and two fully connected ones, that will be the architecture deployed on MNIST-like datasets (explained in Sections 2.1.3 and 2.3 in detail). Even though a VGG network is also a convolutional network, the small convolutional architecture defined below will be referred to as CNN to shorten the description of architecture going forward.

The first convolutional layer filters the 28x28 grey-scale input image with 32 channels of size 3x3 with a stride of 1 pixel (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the max-pooled output (stride 2) of the first conv. layer with a rectified linear unite (ReLu) applied and filters it with double the kernels, 64, again of size 3x3. Then the classifier with two fully connected layers follows, but preceding them are two dropout layers used for regularization.

As [Krizhevsky et al., 2012] explained: dropout layers set outputs of each hidden neuron to zero with a certain probability, usually 0.5. The 'dropped out' neurons do not contribute to the calculation in the forward pass nor in backpropagation. Basically, it simulates as if for every input image, the neural network samples a different architecture, but with all of them sharing weights. Since neurons cannot rely on the presence of particular other neurons, as they get turned off at random during training, they are forced to learn more robust, versatile features. Dropout substantially reduces overfitting.

A dropout layer with a 0.25 chance of dropping a neuron is applied to the flattened output of the second convolutional layer and fed into the first fully connected layer with 1600x128 dimensionality. Followed by a second dropout with 0.5 probability and the second fully connected layer with a dimension of 128x10, for 10 output classes. Depending on the task/dataset the output dimension changes to fit the classification class count. Finally, a softmax function is applied that gives out a distribution for every output neuron.

2.1.2 VGG Networks

VGG Net is the name of the convolutional neural network invented by [Simonyan and Zisserman, 2014] from the Visual Geometry Group (VGG) at the University of Oxford in 2014, that was able to be the 1st runner-up of the 2014 ILSVRC for the classification task. The main improvement compared to prior CNN architectures was archived by increasing the network depth (19 weight layers, 16 convolutional, 3 fully-connected) and by using a configuration with very small (3x3) convolution filters. Even though the depth increased, the smaller filter size incorporates more non-linear rectification, which makes the

decision function more discriminative and reduces the total amount of parameters compared to models with larger filters, e.g. [Krizhevsky et al., 2012].

In regards to [Simonyan and Zisserman, 2014] we use the configuration 'D', commonly known as VGG-16 and presumably the most popular VGG version. The 224x224 RGB image inputs of the network are passed through a stack of convolutional layers, with filters with a small receptive field: 3x3 (the smallest size to capture the notion of direction). The convolution stride is fixed to 1 pixel and the spatial padding of conv. layer input is the same such that the spatial resolution is preserved after convolution. Convolutional layers are organized in blocks with a max-pooling layer at the end, in VGG-16 spatial pooling is carried out by five max-pooling layers with a 2x2 pixel window and stride 2.

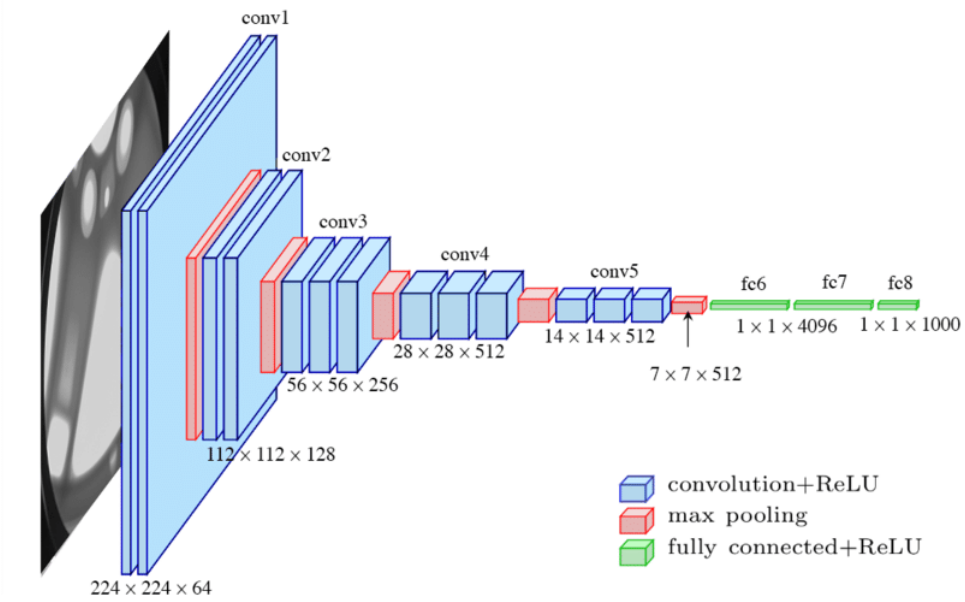


Figure 2.1: VGG-16 network architecture visualization, from [Ferguson et al., 2017]

The number of layers, their dimensions, and feature count can be seen visualized in Fig. 2.1. The first block is made up of two conv. layers with a width of 64 (equals the number of channels) and a max-pooling layer. The dimensions get halved by the pooling, while the channels increase by a factor of 2 in the next block until it reaches 512. The second block contains two 112x112x128 conv. layers. Followed by three blocks each containing three convolutional layers and one max-pooling layer, the first with 256 channels, followed by two with 512 channels. The stack of 16 convolutional layers is followed by a classifier comprised of three fully connected (fc) layers: the first two have 4096 channels each, the thirds channel count depends on the classification task, it needs an output for every class in the dataset, thus in the standard case on ImageNet, it contains 1000 channels (see Section 2.1.3). The final layer is the softmax layer. In accordance with [Krizhevsky et al., 2012], all hidden layers are equipped with the rectified linear unit (ReLu) as non-linearity on

their activations. For training VGG models in the experiments conducted in the following (Section 2.3) also dropout regularisation for the first two fully-connected layers (dropout ratio set to 0.5) in the classifier is used.

2.1.3 Datasets

Fitting the previously discussed network architecture we use datasets similar to the famous MNIST dataset for the small convolutional neural networks of Section 2.1.1. We will refer to them as MNIST-like datasets - constituting of MNIST, FashionMNIST, two MNIST splits, MNIST structured noise, and MNIST noise.

As VGG models take inputs of far bigger dimensionality, the datasets used with the large models of Section 2.1.2 are significantly more complex. They include many of the most widely used datasets in academia and research like ImageNet, Places365, Stanford Cars, VGG-Face, CamVid, and Cifar10 for pre-training and custom3D, Malaria, and Oxford Pet as three different targets.

Table 2.1: Overview and key data of datasets used

Dataset Name	source	target	model	input size	training set	test set	classes
MNIST	✓	✓	CNN	28x28	60,000	10,000	10
FashionMNIST	✓	✓	CNN	28x28	60,000	10,000	10
MNIST Split 1	✓		CNN	28x28	30,000	5,000	5
MNIST Split 2	✓		CNN	28x28	30,000	5,000	5
MNIST noise struct	✓		CNN	28x28	60,000	10,000	10
MNIST noise	✓		CNN	28x28	60,000	10,000	10
ImageNet	✓		VGG	224x224*	1,281,167	150,000	1,000
Places365	✓		VGG	224x224*	1,803,460	346,750**	365
Stanford Cars	✓		VGG	224x224*	8,144	8,041	196
VGG-Face	✓		VGG	224x224*	2,622,000	***	2,622
CamVid	✓		VGG	224x224*	367	334**	11
Cifar10	✓		VGG	32x32	60,000	10,000	10
Custom3D		✓	VGG	224x224*	1,200	240	40
Malaria		✓	VGG	224x224*	22,046	5,512	2
Oxford Pet		✓	VGG	224x224*	5,879	1,470	37

∗ adjusted size for model input ∗∗ incl. validation set ∗∗∗ tested on different dataset (LFW)

In the following, each dataset gets introduced, succeeded by an explanation of additional data augmentations that are applied during training.

MNIST

The MNIST dataset[1] is a modification of the NIST datasets on binary images of handwritten digits by the US National Institute of Standards and Technology. It was created by [LeCun et al., 1998] and still remains as the de facto default benchmark dataset in

[1] http://yann.lecun.com/exdb/mnist/, accessed on 18.12.2020

many machine learning applications, as it is an ideal testbed for new theories and methods.

Consisting of 60,000 training examples, and a test set of 10,000 examples the MNIST database of handwritten digits offers size-normalized and centered 28x28 pixel fixed-size images.

Figure 2.2 below, shows some samples of the MNIST dataset. When using them for learning a classification task with a machine learning algorithm, the pixel values would be inversed, from 0 (black background) to 255 (white digits). This proves to perform better than black on white for two reasons. In general in machine learning, it is good practice to center the data, as with the inverted scales most pixels on the images are black, the mean is closer to 0. Secondly, when updating neural networks the weights corresponding to a 0 in the input are not going to be changed. Intuitively, in this case of centered digits updating weights at the border is not useful, since the corresponding pixels do not discriminate the different digit classes and only the middle ones are changing across samples.

Figure 2.2: MNIST dataset samples from [LeCun et al., 1998]

FashionMNIST

The FashionMNIST dataset[2] was introduced by [Xiao et al., 2017] based on article images from the assortment of Zalando's website.

It was designed to parallel the MNIST dataset in its configuration. Consisting of 60,000 training and 10,000 test samples, as well as 28x28 grayscale images with labels of 10 classes. It is intended to serve as a direct drop-in replacement for the MNIST dataset in benchmarking machine learning algorithms and is very valuable in that, since it poses a more challenging classification task than the MNIST digits data.

[2] `https://github.com/zalandoresearch/fashion-mnist`, accessed on 19.12.2020

The ten different clothing items that make up the classes of FashionMNIST and multiple samples of each are shown in Figure 2.3, highlighting the higher complexity of the dataset.

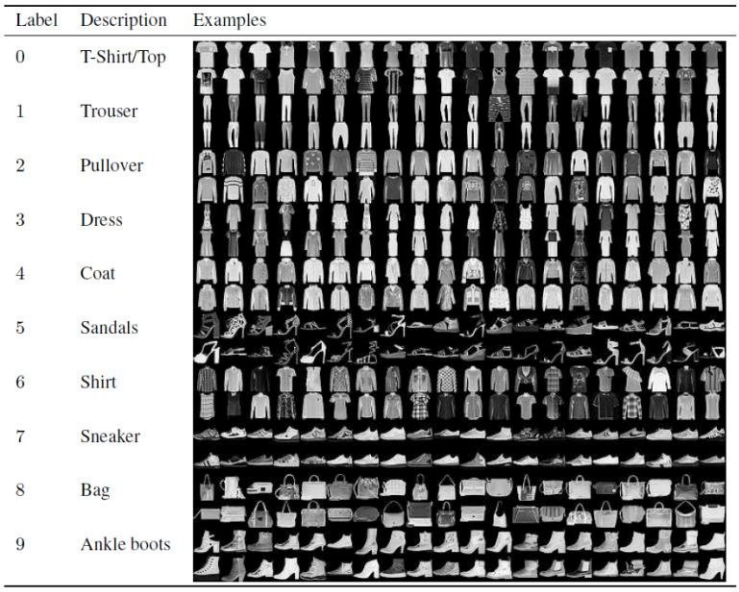


Figure 2.3: FashionMNIST dataset samples from [Xiao et al., 2017]

MNIST splits

To be able to generate results offering a range of performances and differences in representations learned for later comparison, the MNIST dataset is split into two subsets.

MNIST split 1 consists of half the MNIST dataset containing the labels 0, 1, 2, 3, and 4. The second split consists of 5 to 9. Both, therefore, contain 30,000 training and 5,000 test samples.

By constructing the datasets in this way the idea is that each split will still learn good representations for its classes and will consequently benefit the learning as initialization after transfer, but probable with less transfer performance than pre-training on the whole MNIST dataset.

MNIST structured noise

Similar to the MNIST splits, the idea with the MNIST structured noise dataset is to investigate the edge cases in which learning might still work but offers a benchmark at the low end to compare to.

To fit with the architectures and processes the MNIST configurations are maintained, but the 28x28 pixel values are set to random, except for 10% that follow a uniform distribution

fixed for each label. So within mainly random noise, there are 10% of the pixels containing stable information that could be identified to link them to one of the 10 classes. See Section A.1.2 for the detailed implementation.

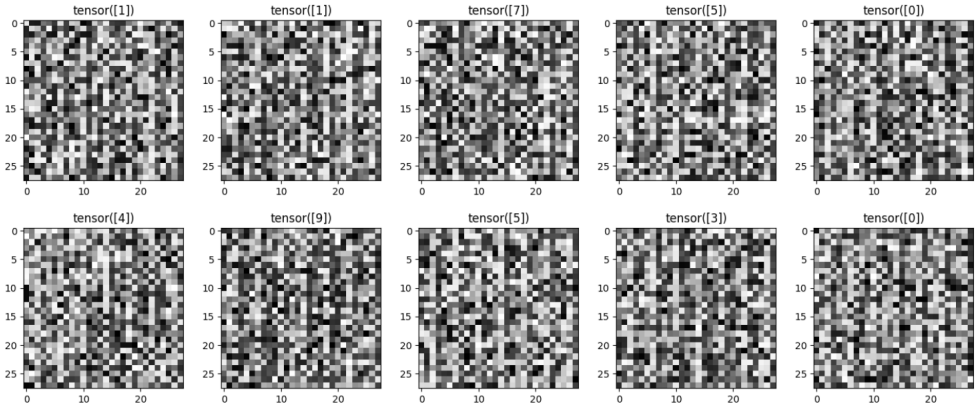


Figure 2.4: Random noise dataset samples

MNIST noise

Like with the structured noise dataset above, the MNIST configurations are kept, but in this case, all pixel values are random. So 28x28 pure noise, should not provide learnable representations and ideally will show a stark contrast in transfer performance by functioning as the lower bound.

ImageNet

The ILSVRC 2012 classification dataset[3], commonly referred to as 'ImageNet', is a subset of the image dataset organized according to the WordNet hierarchy and based on [Deng et al., 2009] of the same name. While the whole ImageNet database contains over 15 million labeled high-resolution images of roughly 22,000 categories, the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) uses only a subset in its annual competition since 2010 [Russakovsky et al., 2015]. After adjusting it in the first two years of the ILSVRC, the 2012 version for classification became the quasi-standard and is used in most machine learning frameworks as 'ImageNet' today.

It comprises 1000 categories and 1,281,167 training samples (with 732-1300 per class) and 150,000 training samples, of which 50,000 are intended for validation.

[3] `http://image-net.org/challenges/LSVRC/2012/index`, accessed on 17.12.2020

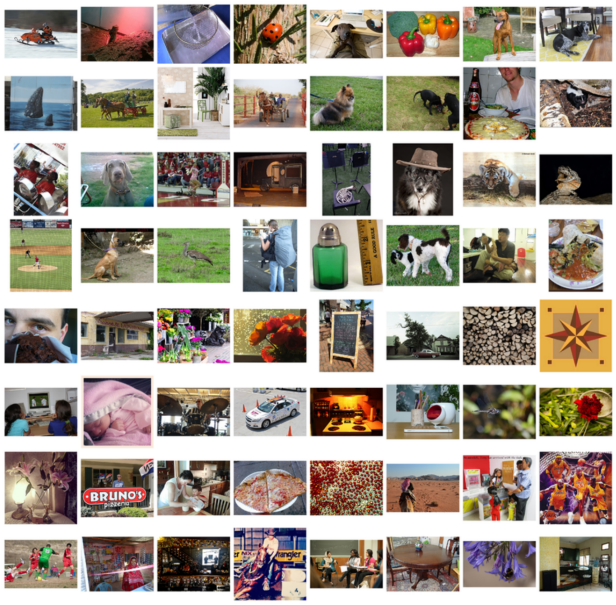


Figure 2.5: ImageNet dataset samples from [Russakovsky et al., 2015]

Places365

The Places365 dataset[4] is a subset of the MIT places database of [Zhou et al., 2017], with images from 365 scene categories.

We use the Places365-Standard version which contains 1,803,460 training images with 3,068 to 5,000 images per class. The validation set has 50 images per class and the test set 900 images per class.

Figure 2.6 displays some of the unique scene categories of Places365's large and diverse list of environment photographs from around the world.

Figure 2.6: Places365 dataset samples from [Zhou et al., 2017]

[4] http://places2.csail.mit.edu/download.html, accessed on 17.12.2020

Stanford Cars

The Stanford Cars dataset[5] contains 16,185 images of 196 classes of cars and was introduced by [Krause et al., 2013]. The dataset is split into 8,144 training images and 8,041 images for testing, with each class roughly split 50-50 for each.

Figure 2.7 shows a sample of all 196 classes of the Stanford Cars dataset. It contains the most popular models in the US since 1990. The Categories typically comprise brand, model, and year, e.g. 2012 Tesla Model S or 2012 BMW M3 Coupe.

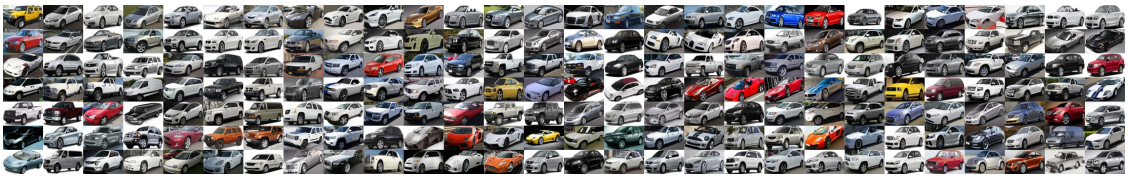


Figure 2.7: Stanford Cars dataset samples from [Krause et al., 2013]

VGG-Face

The VGG Face dataset[6] consists of 2,622 celebrity identities, for each 1,000 different pictures are included, in total 2,622,000 samples. The creation and pre-trained models are based on [Parkhi et al., 2015].

Figure 2.8: VGG Face dataset samples from [Parkhi et al., 2015]

CamVid

The Cambridge-driving Labeled Video Database (CamVid)[7] consists of ten minutes of video footage with pixel-wise annotations for 701 frames into 32 object classes.

As the first collection of videos with object class semantic labels, complete with metadata, the CamVid database addresses the need to evaluate algorithms in driving scenarios. It

[5] https://ai.stanford.edu/~jkrause/cars/car_dataset.html, accessed on 17.12.2020
[6] https://www.robots.ox.ac.uk/~vgg/data/vgg_face/, accessed on 17.12.2020
[7] https://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid, accessed on 28.11.2020

includes over 10 min of high quality 30 Hz footage, with corresponding semantically labeled images at 1 Hz (and one at 15 Hz) [Brostow et al., 2009].

Specifically, there are 5 video clips and for example, one ('seq06R0') consists of 3030 frames at 30Hz, so a total of 1:41 min video footage, and was labeled for 101 frames at 1Hz (matching the total of 1:41 min). Therefore the dataset contains 18,202 frames in total, of which for 701 every pixel is annotated into one of 32 classes with a specific palette of colors, as seen in Fig. 2.9. For training methods of object label propagation are used, given a single frame of labeled objects it propagates those labels through the subsequent frames.

The pre-trained model (SegNet) we use in Section 2.3 is trained on the CamVid dataset according to [Badrinarayanan et al., 2017]. Therefore splitting the labeled frames into 367 training, 101 validation, and 233 testing images at a 360x480 resolution and rougher segmentation into only 11 classes such as road, building, cars, pedestrians, and signs.

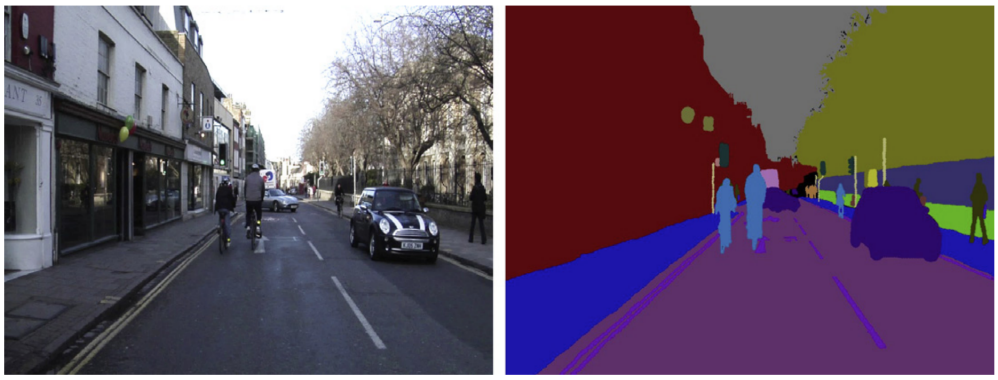


Figure 2.9: CamVid video frame example and corresponding labeled frame from [Brostow et al., 2009]

Cifar10

The CIFAR-10 dataset[8] (named after the Canadian Institute for Advanced Research) is an established computer-vision dataset used for object recognition, introduced by [Krizhevsky et al., 2009].

It consists of 60,000 colour images in 10 classes, seen in Figure 2.10, with 32x32 pixel resolution and 6,000 images per class. It's split into 50,000 training images and 10,000 test images.

[8] `https://www.cs.toronto.edu/~kriz/cifar.html`, accessed on 17.12.2020

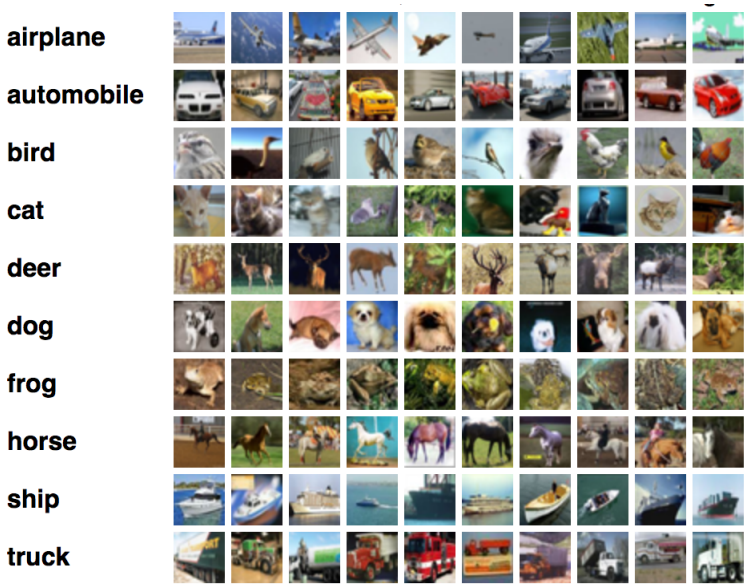


Figure 2.10: Cifar10 dataset classes and samples[8]

Custom3D

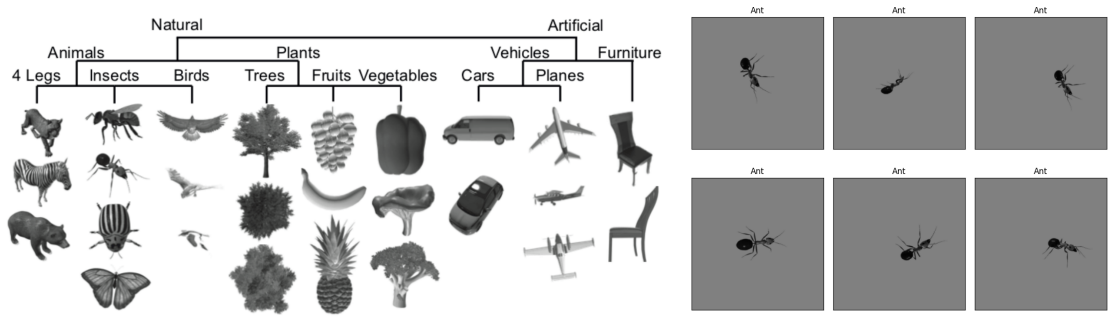
The custom3D dataset[9] is taken from [Ansuini et al., 2019] and is a custom-made dataset of 1400 images developed for the neurophysiological study of [Vascon et al., 2018] containing two dimensional images of 3D objects in different perspectives and illumination.

The dataset consists of 40 three-dimensional computer graphics models, "each rendered in 36 different poses, randomly chosen around four main views (frontal, lateral, top, and 45° in azimuth and elevation), at one of three possible sizes (30-35-40°) chosen at random, in one of three possible positions (0°, ±15°), also chosen at random, and rotated in plane of either 0, 90 or ±45° for a total of 1440 stimuli" [Vascon et al., 2018]. By combining rotations, translations, and size variations the resulting image dataset covers a range of object identities, poses, and additional low-level features e.g. by changing luminance and contrast. Creating a rich and challenging collection while keeping size and complexity within limits and not reaching the scales of other datasets on naturalistic images, like ImageNet.

Using the common training set split of 80% on the 1440 images of the dataset, 30 images for each category or a total of 1,200 are in the training set, while the remaining 6 images for each category (240 total) are used as the test set.

Figure 2.11 (a) shows all 40 classes of the custom3D dataset organized in their semantic hierarchy, while Fig. 2.11 (b) visualizes some of the different perspectives in samples of the 'Ant' class.

[9] https://figshare.com/s/8a039f58c7b84a215b6d, accessed on 17.12.2020



(a) Objects tree structure **(b)** Different perspectives

Figure 2.11: Custom3D dataset from [Vascon et al., 2018]

Malaria

The Malaria dataset[10] contains 27,558 cell images, of two classes: parasitized and uninfected, each with 13,779 images and was published by [Rajaraman et al., 2018].

As Rajaraman et al. motivate: "Malaria is a life-threatening disease caused by Plasmodium parasites that infect the red blood cells. Manual identification and counting of parasitized cells in microscopic thick/thin-film blood examination remains the common, but burdensome method for disease diagnosis. Its diagnostic accuracy is adversely impacted by inter/intra-observer variability, particularly in large-scale screening under resource-constrained settings." [Rajaraman et al., 2018]. Therefore thin-blood smear slides were collected from P. falciparum-infected patients and healthy controls, photographed, and manually annotated by experts to generate the Malaria dataset for use on deep learning algorithms, to aid Malaria diagnosis.

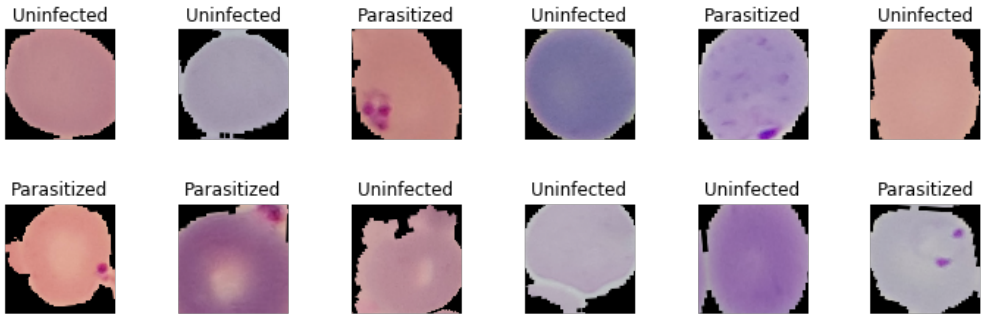


Figure 2.12: Malaria dataset samples

To later fit with our logarithmic scales on the result plots we want the batch size to be a decimal fraction of the training set size. Since we get 22,046 samples with the 80% training split, a batch size of 220 appears to be fitting (same fraction as with custom3D). But without some performance adjustments of the fine-tuning calculations, such a large

[10] https://lhncbc.nlm.nih.gov/publication/pub9932, accessed on 05.12.2020

batch size is already enough to exhaust the memory capacity of the GPU. So like in the CNN runs, a batch is set to a thousandth of an epoch with batch size 22, resulting in two more checkpoints for each run ('_100.pt', '_300.pt') compared to the fine-tuning on custom3D.

Since we use the training set only to fine-tune our pre-trained models, augmenting and sampling them randomly enhances the range in variability that gets learned, but for the test set and especially the extraction of activations for computing the transferability metrics, we want it to be deterministic. That's why we split the data once in the beginning into train and test set and don't apply random transformations, so we can extract the activations for the same 100 samples, 50 per class, on all pre-trained models.

Oxford Pet

The Oxford-IIIT Pet dataset[11] contains images of 12 breeds of cats and 25 breeds of dogs, so a total of 37 different classes with roughly 200 pictures of each. The 7349 images total have large variations in scale, pose, and lighting. It was introduced by [Parkhi et al., 2012] as a challenging fine-grained object categorization benchmark.

The images come in different dimensions, therefore get resized to 224x224 as required for input of the VGG architectures in use, furthermore in contrary to Parkhi et al. an 80/20 split for training and test set is applied, while preserving the number of samples for each class in the splits. Leading to 5,879 training samples and 1,470 test images.

Figure 2.13: Oxford Pet dataset samples

Data augmentation

The most common and simplest method to reduce overfitting on image data is to artificially enlarge the dataset using label preserving transformations. Using the PyTorch 'DataLoader'

[11] https://www.robots.ox.ac.uk/~vgg/data/pets, accessed on 17.12.2020

implementation (from torch.utils.data, see Appendix A.1.2) allows to efficiently load batches of the large datasets and apply transformations to each image.

This form of data augmentation was already applied by [Krizhevsky et al., 2012] and as they explain: allows transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. And since they are generated in python code on the CPU while the GPU is training the model on the previous batch of images, the data augmentations are free in regards to computational time.

The MNIST-like datasets used on the small CNN architecture already come pre-processed in a uniform sample structure, hence the only transformations applied to each sample are a conversion to tensor format and a pixel value normalization to improve model convergence by subtracting the mean from each pixel.

For the datasets on VGG architecture (see Table 2.1) the images come in different shapes with variable resolutions, while the models require ideally a constant input dimensionality. Therefore the RGB images are re-scaled to 256x256x3 by first scaling the shorter side of width or height to 256 pixels and then crop out the central square patch of the other.

During training, the network gets fed with random 224x224 pixel crops from the re-scaled images of the training set, fitting the input dimensionality of the first layer. Since the crops change every time an image is sampled overfitting is reduced by introducing more variance in the training batches (increases the size of the training set by a factor of 2048, though with highly interdependent samples of course). For the test set, a 224x224 center crop is used, which stays fixed.

Besides re-scaling, all samples are transformed to tensors and normalized by subtracting the pixel value mean. The data was further augmented by flipping the image horizontally with a 50% probability. For the Malaria dataset also a random vertical flip transformation, as well as additional slight rotation and color jitter (image data augmentation that randomly changes brightness, contrast, and saturation) are used to artificially enlarge the dataset since it is the least specific with only two classes and blood cell images with sparse visual features. The implementation of the transformation for the Malaria case can be found in the Appendix Listing A.2.

Without this augmentation scheme, the networks would suffer from substantial overfitting, which would have forced the use of much smaller networks.

2.2 Transferability Metrics

In this chapter, three common analytical methods, roughly categorized as geometrical, clustering, and correlation are discussed for applying on activations of deep neural networks and providing quantifiable insight into learned representations.

2.2.1 Intrinsic Dimension (ID)

One way of investigating the workings of deep neural networks is to look at the geometrical properties of the representations they learn in the process of transforming their inputs across multiple layers.

As data is represented by high-dimensional feature vectors, reducing the dimensionality is often a necessity for algorithmic solutions to work in practice. Since in principle it is possible to embed feature vectors in lower-dimensional spaces without loss of information, the question of the minimal number of variables needed to describe the features of a system accurately poses itself. This is known as Intrinsic Dimension (ID), defined by [Bennett, 1969] as the minimum number of parameters needed to generate a data description by maintaining the 'intrinsic' structure characterizing the dataset, so that the information loss is minimized.

Another quite intuitive definition commonly employed comes from the quintessential work on neural networks by [Bishop et al., 1995]: "a set in d dimensions is said to have an ID equal to d' if the data lies entirely within a d'-dimensional subspace."

In the following we study the intrinsic dimension as a fundamental geometric property of data-representations in neural networks, i.e. the minimal number of parameters needed to describe a representation, based on [Ansuini et al., 2019]. As they further state: "It is widely appreciated that deep neural networks are over-parametrized, and that there is substantial redundancy amongst the weights and activations of deep nets - e.g., several studies in network compression have shown that many weights in deep neural networks can be pruned without significant loss in classification performance."

As research on dimensionality reduction and intrinsic dimension progressed, more and more estimators for large, complex data manifolds were developed. One of the approaches, the Nearest Neighbors-Based ID estimators, describes the distributions of data neighborhoods as functions of the intrinsic dimension d, assuming that close points are uniformly drawn from small enough d-dimensional hyperspheres. Building on this premise [Facco et al., 2017] introduced 'TWO-NN', an ID-estimator that only uses the distances to the first two nearest neighbors of each point, since "this minimal choice for the neighborhood size allows to lower the influence of dataset inhomogeneities in the estimation process" [Facco et al., 2017]. Therefore the procedure is computationally more efficient than previous methods (see Fig. 2.14 for details). Also, TWO-NN can be applied in many challenging circumstances

like on manifolds containing the data that are curved, topologically complex, or sampled non-uniformly.

This makes [Ansuini et al., 2019] choose TWO-NN as they set out to characterize how the ID of data representations vary across the layers of CNNs and how it relates to generalization for the first time. Especially since the TWO-NN method of computing ID allows this kind of examination to be conducted efficiently for data set with $O(10^4)$ samples, each with $O(10^5)$ coordinates (as activations in an intermediate layer of a CNN).

Utilization: Inspired by the capability of TWO-NN to map out ID across multiple layers and networks and the findings of Ansuini et al. showing the ID to be: orders of magnitude smaller than the number of units in each layer, following a similar trend along the hidden layers of CNNs of first expanding dimensionality and then monotonically decreasing to low values in the final layers, as well as the existence of a connection between ID and generalization during training, we use ID based on TWO-NN in this thesis as well.

For one ID is capable of generating useful insight into transfer learning on deep neural networks due to its investigation of geometrical properties but especially since "in networks trained to classify images, the ID of the training set in the last hidden layer is an accurate predictor of the network's classification accuracy on the test set" [Ansuini et al., 2019] was shown and empirically validated on multiple networks by Ansuini et al., it's a potential metrics fitting our goal of predicting transfer performance.

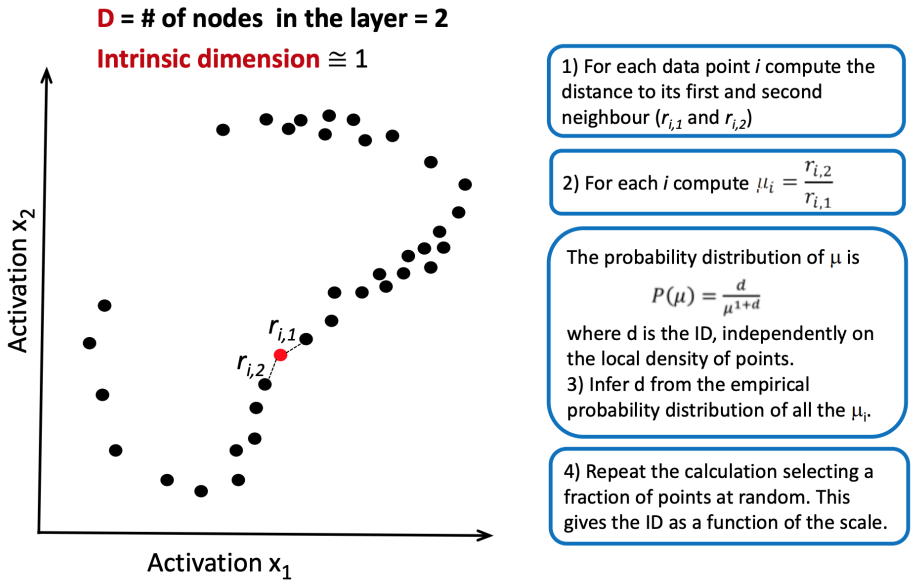


Figure 2.14: TWO-NN estimation procedure: estimating ID from the statistics of nearest neighbour distances, from [Ansuini et al., 2019]

2.2.2 Sum of Squares (SS)

Clustering analysis, basically the task of grouping a set of objects in such a way that objects in the same cluster are more similar to each other than to those in other clusters, is a fundamental tool in machine learning and has been widely applied in various domains. One commonly known branch of classical clustering contains the methods of K-Means [MacQueen et al., 1967] and Gaussian Mixture Models [Bishop, 2006], which are fast, easy to understand, and can be applied to a large number of pattern recognition problems, such as unsupervised learning.

Using Euclidean distance and sum of squares one can investigate the variance within data of the same class, belonging to one cluster, as well as the distance of their centroids to other clusters, by using the decomposition of Analysis of Variance (ANOVA).

The within-cluster variance and between cluster variance can be calculated by the sum of squares within (SSW) and sum of squares between (SSB) respectively. Let $X = x_1, x_2 \dots x_n$ be a set of data with n samples. Suppose the samples in X have labels of m classes that mark them as representatives of non-overlapping clusters $C = C_1, C_2 \dots C_m$, with centroids $c_1, c_2 \dots c_m$.

The value of SSW is defined as:

$$SSW(X) = \sum_{i=1}^{m} \sum_{j \in C_i} \|x_j - c_i\| \tag{2.1}$$

which is minimized over all partitions C in clustering methods and gives a description of how well a model representation fits for labeled data. According to ANOVA, the total sum of squares (TSS) can be decomposed into two parts, that are SSW and SSB for any partition [Fisher et al., 1921]. Leading to

$$SSB(X) = \sum_{i=1}^{m} |C_i| \, \|c_i - \bar{x}\| \tag{2.2}$$

where $|C_i|$ is the number of elements in each cluster, and $\bar{x}$ is the mean value of the whole data set.

And therefore the total sum of squares can be expressed by both the following statements

$$TSS(X) = \sum_{i=1}^{n} \|x_i - \bar{x}\| \; = SSW(X) + SSB(X) \tag{2.3}$$

Utilization: Intuitively a model that minimizes the within-class variance, i.e. identifies samples of the same class as being close, while clearly separating between class clusters, learned a good representation, and is capable of performing well in classification. Therefore we use cluster analysis applied to activations across the layers of different networks to

quantify their learning performance, by calculating the ratio SSW/TSS, which expresses the quality of clustering by relating the minimizing of the within sum of squares to the total. This is chosen as the second metric, in the following abridged to sum of squares (SS), to investigate the relation to transfer performance through clustering.

2.2.3 Representational Similarity Analysis (RSA)

Originating in the field of computational neuroscience, Representation Similarity Analysis (RSA) was introduced by [Kriegeskorte et al., 2008] and since then became a widely used data-analytical framework for quantitatively relating brain activity measurements with computational and behavioral models.

However, in recent years RSA also started to gained traction for its application on deep neural networks, providing insight into the workings of artificial neural networks similarly to human brains in neuroscience. Only recently, in the works of [McClure and Kriegeskorte, 2016] and [Mehrer et al., 2018] the approach of RSA was applied to assess the similarities of two purely computational models. Before it had always been the relation between computational model and brain data.

[Dwivedi and Roig, 2019], for the first time, introduced RSA as a similarity measure for relationships between tasks in regards to transfer learning. Together with their recent work on a measure in [Dwivedi et al., 2020], it shows big potential for RSA in research on deep learning and computer vision and motivated its use as the third major metric in our cause.

In RSA, measurements are related by comparing their representation-activity in dissimilarity matrices. These dissimilarity matrices are obtained by comparing pair-wise correlations between activities/representations associated with each pair of inputs. For the application on DNNs, this means that a set of image samples is selected as inputs and by performing a forward pass for each sample the representations (activations), on each layer, are collected. For each pair of inputs, the Pearson's correlation coefficient ρ is calculated and the desired dissimilarity score is obtained by $1 - \rho$.

These dissimilarity values for each pair of inputs then form the symmetric representational dissimilarity matrix (RDM). And while [Dwivedi and Roig, 2019] continue to follow the process of RSA by then taking the Spearman's correlation between different RDMs, we look at the different parts of the RDM when sorted for its inputs.

Utilization: While the diagonal of a RDM always is 0, since the dissimilarity of an input with itself is 0 (as $\rho = 1$), the block structure obtained, if the inputs of the matrix are sorted by their labels, shows the within-class dissimilarity. Which should be minimized, while high dissimilarity to the other classes should intuitively mean a clearer separation and stronger classification learned in the representation.

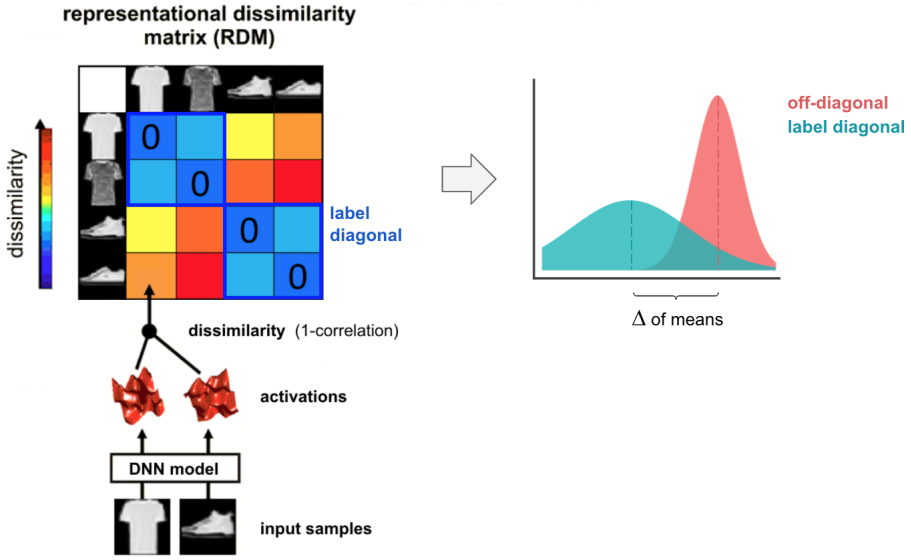


Figure 2.15: RSA: calculation of RDM and delta of the mean diagonal values

See Figure 2.15 for a simplified illustration with 2 classes of the FashionMNIST dataset - shirts and shoes. While the diagonal is 0, we call the block-diagonal, with a size of the sample count per class, the 'label diagonal' and the other RDM entries 'off-diagonal'. To quantify how well a model is able to separate inputs of different classes in its representations we take the dissimilarity values of the RDM and follow the assumption that all label diagonal elements should be close to 0 while having higher dissimilarity to off-diagonal elements.

Therefore, we take the mean of within-class dissimilarities (label diagonal) - excluding the strict diagonal values to not skew the distribution - and the mean for off-diagonal and calculate the delta between them (as seen on the right of Fig. 2.15). This delta will function as the RSA metric, since for untrained models the distance between the two means is low or non-existent, while it widens with rising generalization performance during training.

2.3 Experimental Setup

To investigate transferability of the architectures defined at the beginning of the chapter in Sections 2.1.1 and 2.1.2 for classification tasks on the datasets described in Section 2.1.3, experiments are conducted to gather empirical evidence.

Therefore in the following Section 2.3.1, pre-training of all the models is explained, followed by extracting their activations when fed target task samples and applying the metrics in Section 2.3.2. Finally, the transfer and fine-tuning on the defined target tasks is explained in Section 2.3.3.

Similarly to [Zamir et al., 2018] we set up our empirical experiments as training multiple fully supervised task-specific models, with identical architecture (besides output dimensionality), hyperparameters and training samples. Then we transfer the weights of the pre-trained models to fine-tune them on the target tasks and derive the fine-tune performance which is used as ground-truth in evaluating the transferability metrics. Contrary to Zamir et al. transfer is not investigated between each task, but we set MNIST and FashionMNIST as target tasks on the CNN architecture and the datasets custom3D, Malaria, and Oxford Pet as targets for VGG. Meaning every pre-trained model gets transferred and fine-tuned on these target datasets. For fine-tuning, the same architecture and hyperparameters as for the task-specific networks are used, except that the learning rate gets anneal since they train much faster.

The implementation is based on the PyTorch machine learning framework and the CUDA toolkit for GPU calculations. All the code for replicating the experiments, as well as the evaluation and visualization can be found on GitHub, see Appendix Section A.1.2. Also for an additional overview of the experimental process see Fig. A.1 in the Appendix as well.

2.3.1 Pre-training

Models trained on the six different classification tasks are needed for the MNIST-like datasets on the small CNN architecture and another six for the VGG models.

Pre-training on CNN architecture: Since random initialization can have huge impacts on the learning performance of neural networks (see [Mehrer et al., 2018]) and obtained results should be checked for statistical significance, not just one model gets trained for every task, instead of for 10 different seeds a total of 60 training runs are conducted.

During each of the training processes, multiple checkpoints are saved to analyze the importance of pre-train duration later on. Checkpoints are taken on a logarithmic scale, fitting with the steep accuracy increase on the relatively simple MNIST-like datasets, as well as with the later evaluation and visualization in mind, that will reveal more information

on the log-scale. Batch size is chosen to be fractions to the power of 10, i.e. one batch has 64 samples which are roughly $\frac{1}{1000}$ of the 60,000 training samples encompassing one epoch. Checkpoints are set from 10^{-3} to 10^{2}. Added in between these is one more checkpoint roughly half-way on the log-scale to gain greater resolution. In total, the checkpoints during training are taken after 1 batch, 3 batches, 10 batches, 30 batches, 100 batches, 300 batches, 1 epoch, 3 epochs, 10 epochs, 30 epochs, and 100 epochs. Additionally, the untrained model at epoch 0 is also saved - giving a total of 12 checkpoints for every run. For the 10 seeds for every task, this amounts to 120 checkpoints for which the model state is saved, as well as training and test accuracy and loss. Therefore a total of 720 checkpoints are taken for pre-training the six tasks on the CNN architecture.

As explained in Section 2.1.3 and shown in Table 2.1, the MNIST split datasets only contain a half of the MNIST dataset. Hence the model architecture is adjusted on the output layer dimensionality to 5, according to their class count, but otherwise remains unchanged. Also to fit with the scale for the early checkpoints the batch size is set to 32.

A learning rate of 0.0001 is used with the Adam optimizer from [Kingma and Ba, 2014] and cross-entropy to calculate the loss. Training duration is set to 100 epochs. Longer training durations were also tried, but they provided only marginal improvements in accuracy if any, after 100 epochs the performance stayed mostly stagnant or even decreased slightly.

Pre-training on VGG architecture: The very deep VGG architecture and the large-scale datasets applied to it take tremendous computational time and power to train. As [Simonyan and Zisserman, 2014] states: "On a system equipped with four NVIDIA Titan Black GPUs, training a single net took 2-3 weeks depending on the architecture." Since our interest is focused on transferring and not archiving state-of-the-art accuracy performances, we use publicly available pre-trained models for the VGG architecture and concentrate our research and computational efforts on transferring and fine-tuning them on the three target datasets.

ImageNet: For the ImageNet dataset obtaining pre-trained weights is very easy for most common architectures, since most deep learning frameworks support loading models with weights pre-trained on ImageNet already. Pytorch does so too and the pre-trained model state included in the torchvision library[12] trained on the ILSVRC 2012 classification dataset is used for the VGG-16 model.

Places365: Weights for a VGG-16 model pre-trained on the Places365 dataset can be found on the Github page[13] of the authors of [Zhou et al., 2017]. Since it is only available

[12] `https://pytorch.org/docs/stable/torchvision/models.html#vgg16`, accessed on 10.11.2020
[13] `http://places2.csail.mit.edu/models_places365`, accessed on 08.12.2020

as a caffe model, caffe is another ML framework, one needs to convert the structure of the model state into PyTorch format. The 'caffemodel2pytorch' package[14] makes it easy to load the model as a caffe model (loads caffe prototxt and weights with PyTorch as backend) and allows to save the weights as a state dictionary, which can then be loaded and assigned layer-wise to a PyTorch model.

Stanford Cars: Again weights for a VGG-16 model pre-trained on the Stanford Cars dataset can be found on GitHub[15] but this time in the format of the MxNet framework. Conversion is needed and archived by utilizing the comprehensive open-source cross-framework tool MMdnn[16] from Microsoft.

VGG-Face: The weights to the VGG-16 model in [Parkhi et al., 2015] can be found on Samuel Albanie's Oxford VGG website[17].

CamVid/SegNet: Since SegNet, [Badrinarayanan et al., 2017], uses an encoder network that is topologically identical to the 13 convolutional layers in a VGG network and we are going to newly initialize the fully connected classifier layers anyways in fine-tuning and the main focus for activation extraction is on the convolutional layers, we can use the encoder weights as a pre-trained model for the CamVid dataset. Even though there are differences in the architecture, because the VGG-16 configuration with batch-normalization is used, we included CamVid to diversify the task range and to build upon work conducted at the NI chair at TU Berlin. The weights are derived from the encoder layers of SegNets trained by Heiner Spiess for a previous thesis with the chair.

Cifar10 For Cifar10 weights can be found on GitHub[18], unfortunately only for VGG-19 architecture. But the only difference is one more convolutional layer in the later convolution blocks, therefore we can also use this configuration with some small adjustments. Since the size of Cifar10 images is quite small with only 32x32 pixels, compared to the standard 224x224 on VGG, one has to pay close attention in regards to the pooling and adjust the classifier layers. The further into a VGG model the more the original input size gets shrunk down, a total of 5 max-pooling layers with stride 2 each half the input size. Therefore at the end of the feature extractor part, i.e. after the 5th convolutional block, the size of Cifar10 inputs remains only at 1x1x512 (compared to 7x7x512 for 224x224 inputs, see Fig. 2.1). Instead of a linear layer with input 25,088 and output 4,096, the fully connected classifier layers are adjusted to input and output size 512 and final output 10 accordingly.

[14] `https://github.com/vadimkantorov/caffemodel2pytorch`, accessed on 08.12.2020
[15] `https://github.com/afifai/car_recognizer_aiforsea`, accessed on 08.12.2020
[16] `https://github.com/microsoft/MMdnn`, accessed on 08.12.2020
[17] `https://www.robots.ox.ac.uk/~albanie/pytorch-models.html`, accessed on 08.12.2020
[18] `https://github.com/geifmany/cifar-vgg`, accessed on 08.12.2020

Baselines: On the CNN architecture, MNIST and FashionMNIST are set as both, source and target tasks, which comes in handy to use their pre-training performances also as a baseline for the evaluation of transfer later on. Since under omission of data restrictions in the target domain, one could just train a random initialized model on the target task, without using transfer learning.

For VGG the target tasks are chosen to be 'small' and not comprise training sets with millions of samples, therefore the baselines can be calculated with manageable resources. That means we pre-train VGG models for each of the target tasks as benchmarks on custom3D, Malaria, and Oxford Pet and doing so again for multiple seeds. Stochastic gradient descent with momentum 0.9 is used as optimizer, with a learning rate scheduler reducing the starting rate of 0.01 after every 20 epoch by factor 0.1. For custom3D the batch size is set to 12 and for Oxford Pet to 58, to fall in the range of computationally possible fractions of the epoch size. Since then one batch only equals roughly a hundredths of an epoch, in the training runs for these two targets there are two fewer checkpoints (at 100 batches and 300 batches). For training the classification task on the Malaria dataset a batch size of 22 is chosen, there it is in accordance with the above-explained checkpoint structure, like on CNN. Two dropout layers are used for regularization as explained as part of the VGG architecture in Section 2.1.2. The total training duration again is set to 100 epochs.

Additionally, we save multiple untrained, random initialized models for each case as another baseline.

2.3.2 Extract Activations

Since our main objective is applying the metrics to gain insight into transferability performance by quantifying representations learned, gaining the data of these representations by saving the activations of each layer of the pre-trained models is integral.

We implement that in PyTorch by setting a hook for the outputs of each layer and save the activations as tensors. For CNN models we do so after every layer, while on the VGG architecture we save the activations after every pooling layer, so once for every convolutional block, as well as for the fully connected layers.

Since we want to make a prediction regarding the target task, a few samples from the target domain are fixed and fed into the pre-trained models, which were trained on a different task. For each sample, a forward pass through the model is calculated and the activations saved from the selected layer outputs. For MNIST and FashionMNIST, the datasets on CNN, 500 samples with equal count per class are set for extracting activations. While on VGG, 1,200 custom3D samples, 30 per class, are taken for activation extraction, and also for the Oxford Pet dataset 30 samples per class are selected, a total of 1,110. Whereas for Malaria again 50 samples per class are chosen, since it only contains two classes, and the

activations for 100 samples are saved. Even if these subset sizes for extraction seem small, due to the number of models and layers they still quickly accumulate to gigabytes of data, therefore are set to the maximum scope of our capabilities already.

Since for CamVid and Cifar10, the architectures are slightly different the extract functions have to be adjusted to work with the additional layers. In Appendix Section A.1.2 the code for extracting the activations from the standard VGG-16 model can be found (Listing A.3).

On the extracted activation tensors we apply the three metrics defined in Section 2.2. Meaning for Intrinsic Dimension the high dimensional activation tensors get analyzed for their smallest subspace embedding, while for each sample and its label the SS metric performs clustering and distance calculation between the tensors and RSA derives Pearson's correlation for each pair of tensors to put into RDMs and calculate the mean distance for label- and off-diagonal. The results obtained from the extracted activations and the metrics are discussed in the next chapter.

2.3.3 Fine-tuning

Once we gathered pre-trained models for all the tasks and extracted the activations for samples of the target tasks to calculate the transferability metrics, we need a way to evaluate their predictive performance. To do so, the ground-truth is needed to compare how well a prediction aligns with the actual outcome, e.g. by Spearman's rank correlation. For this reason, we actually transfer and fine-tune each model on the target tasks. If our metrics perform well, this trial and error computation could be avoided, which motivates our evaluation on the practical level.

For each checkpoint of the pre-trained models with CNN architecture we load the model state but only keep the weights of the convolutional layers and substitute the fully connected layers with newly initialized ones. Also adjusting the last layers output dimensionality to fit the MNIST and FashionMNIST target tasks. Otherwise, the model architecture and parameters stay the same. Even the training parameters are mostly analog to the CNN pre-training (see Section 2.3.1), training for 100 epochs with a batch size of 64 and learning rate 0.0001, using Adam and cross-entropy loss. During fine-tuning, we train all weights on CNN, the new ones of the fully connected layers as well as the transferred convolutional ones.

Meaning a total of 120 models in each of the CNN pre-train cases (10 seeds x 12 checkpoints) gets fine-tuned for evaluation.

For the pre-trained VGG models, the feature extractor layers get transferred and a newly initialized classifier is added, with output layer dimensionality fitting the class count of the target tasks custom3D, Malaria, or Oxford Pet according to the architecture laid out in Section 2.1.2. For this architecture the transferred weights get frozen, all but the last convolutional one before the classifier. So only the fully connected layers and the last

convolutional layer get their weight parameters actually fine-tuned. This method is inspired by the fine-tuning process in [Ansuini et al., 2019].

As in the pre-training of the VGG models (see Section 2.3.1) stochastic gradient descent with momentum 0.9 is used with a learning rate scheduler reducing the starting rate of 0.01 after every 20 epoch by factor 0.1. Also, the batch sizes are 12 for custom3D, 58 for Oxford Pet, and 22 for Malaria again, as are the other hyperparameters. In the special cases for CamVid and Cifar10 the slightly different architecture of the pre-trained models with additional layers is carried over to the fine-tuned model in the target domain. Additionally, an untrained random initialized model gets fine-tuned in the same way to provide a benchmark in the subsequent analysis of the results.

3 Results

In this chapter, all the results of the conducted experiments, meaning the training of neural networks, the extraction of activations and calculation of the metrics, as well as the actual transfer of weights and fine-tuning on new target tasks, as described in the experimental setup (Chapter 2.3) are visualized and discussed. The main objective is the verification that the three metric candidates defined in Section 2.2 offer statistically valuable insight into the fine-tuning performance after transfer learning.

The results retrieved are split up into Section 3.1 for all the findings in regard to the CNN architecture and Section 3.2 for the experiments on VGG models. For both first, the pre-training on the source tasks is discussed, followed by the evaluation of the metrics calculated on the extracted activations on each layer and finally the outcomes of actually transferring and fine-tuning the models on the target tasks.

Finally, in Section 3.3 Spearman's rank-correlation is applied on the metrics scores and the 'ground-truth' fine-tuning performance on the target task to evaluate how capable each of the metrics is to quantify transfer performance, before transferring.

3.1 Results for CNN architecture

3.1.1 Pre-training

As described in Section 2.3.1 we pre-train 10 different CNN models based on 10 different seeds for the random initialization of the model weights on several source tasks and take 12 checkpoints during the training process.

Since the process is conducted 10 times for different seeds to avoid strong influences on results by the starting weight initialization, in Figure 3.1 and most subsequent plots and analysis the mean is taken of the same model for the different seeds. Visualizing the accuracy archived by all of them grants the results seen in Fig. 3.1.

The choice for logarithmic scale on the x-axis and for setting the training checkpoints (1 batch equals roughly 0.001 epoch) allows to visualize the early learning improvements, otherwise, there would hardly be visible differences on a linear scale. It is clearly visible that the CNN model architecture is capable of learning all source tasks, especially the MNIST and noise ones are fairly easy, with accuracies over 98% already after 3 epochs. FashionMNIST shows its purpose of a more complex benchmark as it continues to rise in

performance accuracy up to 100 epochs, but also reaches a high value of over 90%. As was the reason for including the pure noise case, it stays stagnant at the very bottom (it still has 10 labels, therefore worst performance is always 10% accuracy) and works as the bottom benchmark. Surprisingly, the learning for the structured noise dataset is delayed but the model is capable of learning the slightly added regularities within the noise, even though it tends to overfit in the end.

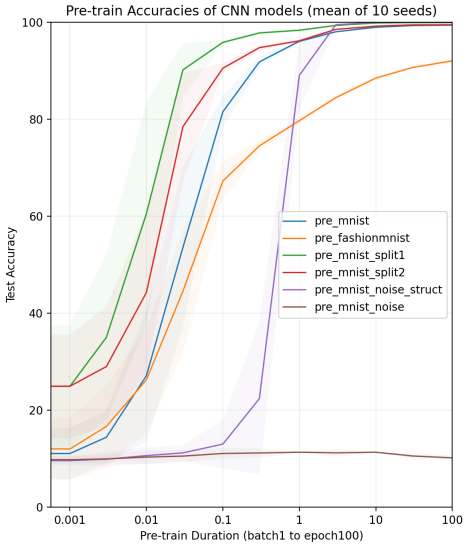


Figure 3.1: Pre-train Accuracy on CNN models

3.1.2 Evaluate metrics

The activations of all the pre-trained models when 500 samples of the target tasks MNIST and FashionMNIST are fed in are taken on every layer and the ID, SS, and RSA metric for all of them are calculated.

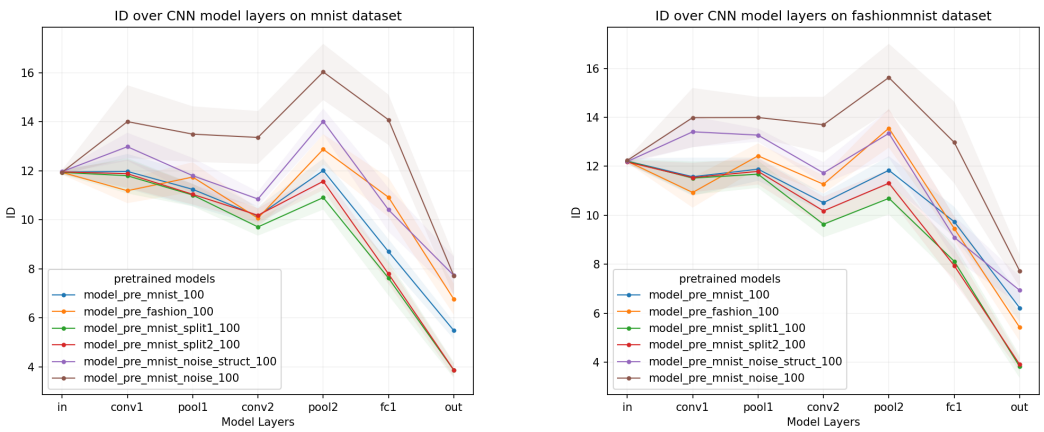
Since the same target dataset subset is used each time for the activation extraction on every model, the activation of the input layer, i.e. the raw pixel values before any processing, is of course the same for each metric. As it's also proven, that the fully connected classifier layers are already too specialized to benefit the transfer performance [Yosinski et al., 2014] and it's common practice to only transfer prior layers, the focus for analyzing the results is on the layers 2 to 5, meaning both the convolutional and the pooling layers. In the following the results for these layers are discussed and especially of the last feature extractor layer 'pool2', since it is the one up to which weights get transferred most commonly and its output gets fed into the new model continuation in the target domain which gets fine-tuned.

The following Figures 3.2, 3.3 and 3.4 show the results of each of the three metrics on the pre-trained models for the two targets MNIST and FashionMNIST next to each other.

The activations are extracted on the longest pre-trained models (100 epochs) in each case.

The shade around each line plot gives the 95% interval, approximated by $\pm 2 * std$ calculated over the metrics of 10 different models for the 10 seeds. As one can see the scattering is highest for the models pre-trained on the noise dataset (on the feature extractor layers) as was expected since it should have learned the weakest representations if any, implying the biggest variance in values for the metrics. Also fitting with intuition are the tighter deviations for the models pre-trained on MNIST and its two splits on MNIST itself as target, while pre-training on FashionMNIST shows the least scatter for itself as target.

Interestingly, for both targets, all the results of intrinsic dimension in Fig. 3.2 follow the same trend, but there are clear differences between highest and lowest values on layer 'pool2', e.g. for MNIST as target the ID metric of pre-training on MNIST split 1 is 10.9 while on MNIST noise the value is 16.1 so a 48% difference. For all three pre-train datasets, MNIST and its two subsets, the values are close throughout the model depth, deviating on 'pool2' only from 10.9 to 12. Unfortunately on FashionMNIST as target the analog, that FashionMNIST itself would be the best pre-training for the metric score, does not hold, instead the MNIST datasets again have the lowest score on 'pool2', therefore are best capable of reducing their representations for FashionMNIST samples to a smaller dimensionality.

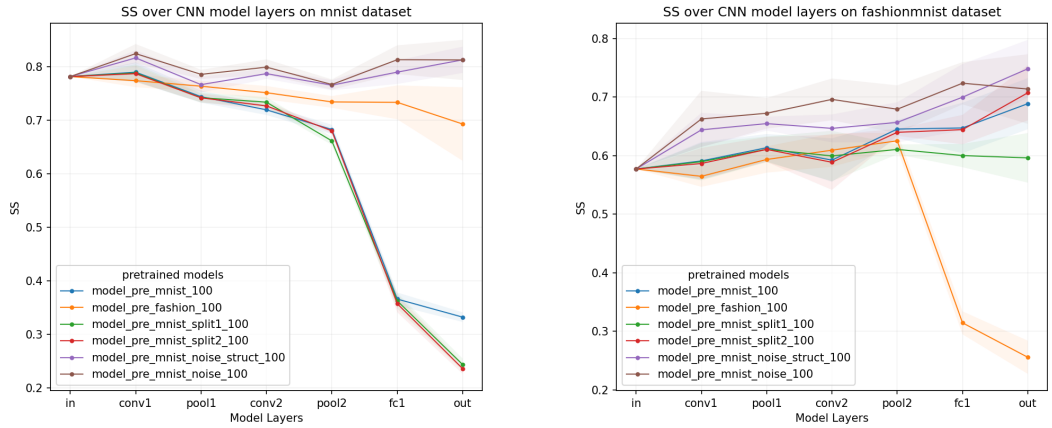


(a) Target MNIST **(b)** Target FashionMNIST

Figure 3.2: ID metric results for CNN models

By design of the convolutional neural network architecture, the parameter count first increases with applying the convolutional kernel multiple times and gets reduced in later layers through pooling, therefore a reduction in ID deeper into the model is by design always going to happen. Hence the results for the pure noise model also decrease in both SS plots, while they stay stagnant for the other metrics in the following.

Very clearly the plot of the sum of squares metric in Fig. 3.3 shows which models are capable



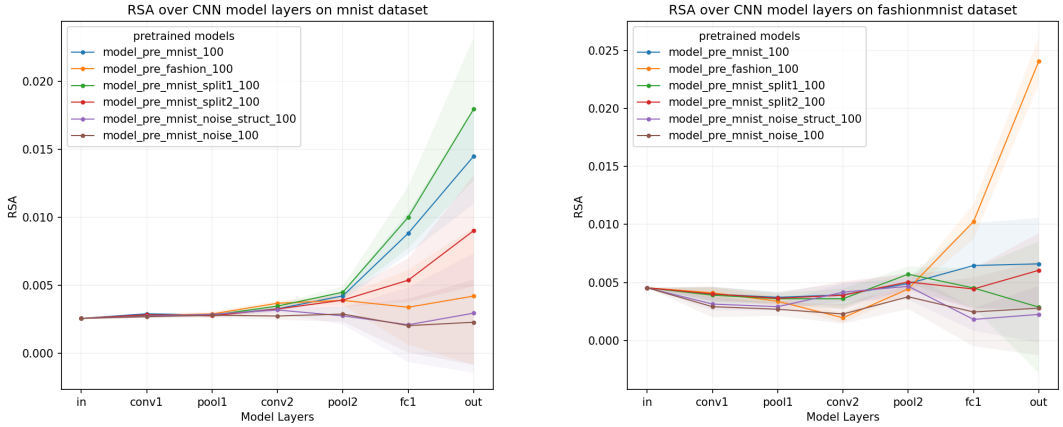
(a) Target MNIST **(b)** Target FashionMNIST

Figure 3.3: SS metric results for CNN models

of clustering the target task samples well. For MNIST as target, itself and its two splits clearly outperform the other pre-trained models. Especially visible is it for this metric on classifier layers, since its fully connected layers are highly specialized for assigning MNIST features to the MNIST classes on its output. But also on layer 'pool2' the MNIST and MNIST splits models perform better, with FashionMNIST in-between and structured noise and noise as expected worse and stagnant throughout the layers.

On FashionMNIST as target the separation is not as apparent, it stays within the deviation interval of other models, and only on the classifier layers it also shows clearly the best scores.

While for the RSA metric (Fig. 3.4) the differences between the label diagonal correlation value mean and the off-diagonal one are very similar at first, on the 'pool2' layer the models pre-trained on non-noise datasets start to increase. This trend is amplified in the last two



(a) Target MNIST **(b)** Target FashionMNIST

Figure 3.4: RSA metric results for CNN models

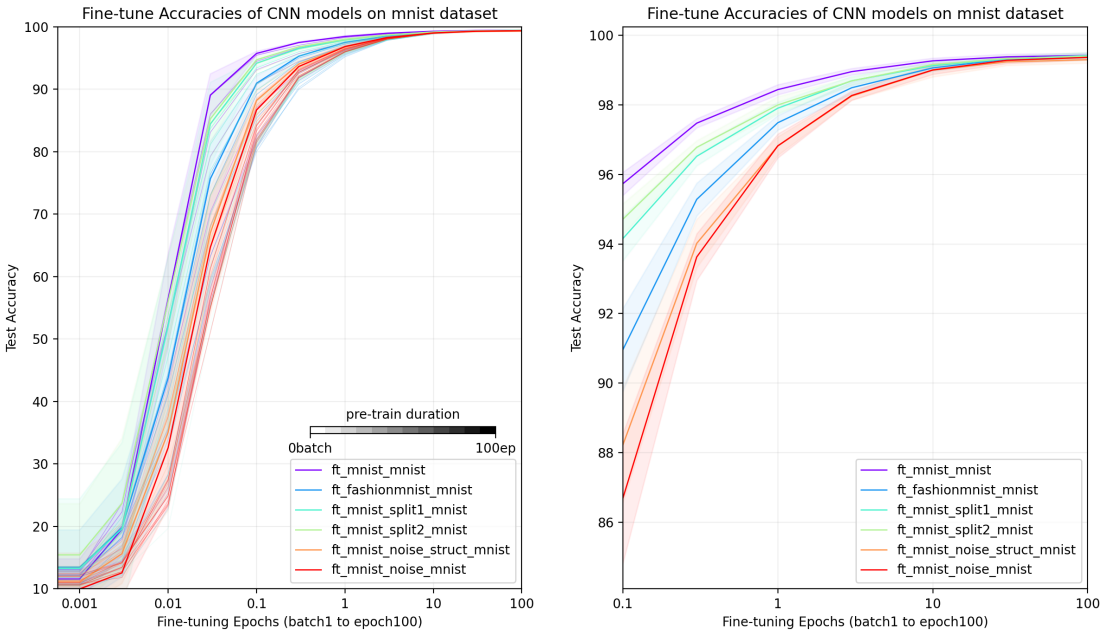
layers, but already on 'pool2' worst and best metric value show a 57% difference, even on a smaller scale. Also, they are statistically significant since deviations are controlled by non-overlapping 95% intervals.

For FashionMNIST as target models pre-trained on itself also clearly outperform on the last two layers, but don't show strong separation before.

3.1.3 Transfer learning outcomes

Transferring the weights of the pre-trained models to the target task according to the process described in Section 2.3.3 and adding a new classifier, a model in the target domain is gained, of which all weights are set to be fine-tuned and the following results obtained by doing so for the two target tasks MNIST and FashionMNIST.

Figure 3.5 shows all 72 models (6 datasets x 12 checkpoints) in subplot (a) with a color lucency gradient for the checkpoints from untrained (batch 0) to 100 epochs, the latter being the thick line in strong color. The lines of earlier pre-train checkpoints in lighter color show the development and influence of the pre-training duration at each fine-tuning checkpoint (x-axis, and again in log scale).



(a) MNIST fine-tuning results for all checkpoints (pre-train duration as color gradient)

(b) Detail view of later results for 100 epochs pre-trained models

Figure 3.5: Fine-tuning results for CNN models on MNIST

The visualization next to it deliberately discards the earlier checkpoints and only shows the results for transferring and fine-tuning the models pre-trained for 100 epochs and additionally does so for the later fine-tuning epochs, to see more detail than before. On the other hand, even in the detailed few the difference between the models is down to less

than a fraction of an accuracy percentage point after 10 epochs of fine-tuning. This is due to the relative simplicity of the classification tasks on MNIST like datasets, but highlights the decision for including early batch learning performance in the analysis, since e.g. after 30 batches fine-tuning the difference between the longest and shortest pre-trained models is roughly 40 accuracy percentage points. In an actual application of transfer learning the target domain is most likely data restricted (due to labeling cost and availability), therefore the early batches give some insight into how training with limited data would presumably benefit most. Also, it motivates the inclusion of much more complex datasets and tasks on the deeper VGG networks of the other experiments, to see results spanning the whole range of accuracy results.

To further investigate the influence of pre-training on the transfer performance on the target task MNIST it is interesting to look at the difference of the outcomes in regard to the baseline case of just training a model on MNIST from scratch. This is visualized in Figure 3.6, where the accuracy of the baseline is subtracted from all the results of Fig. 3.5 (a), putting the results for just training MNIST on the flat 0 axis of the delta plot and showing how much each of the transfer cases deviates from that. Additional the standard deviation of the baseline case (over the 10 different initialized seed models) is plotted with the grey shade, showing that the transfer results for noise and structured noise lay within the deviation of normal training most of the time. But even clearer than before, the exceeding performance of the other non-noise models as initialization for learning MNIST in earlier batches of fine-tuning is visible, as well as the fading influence at higher training duration.

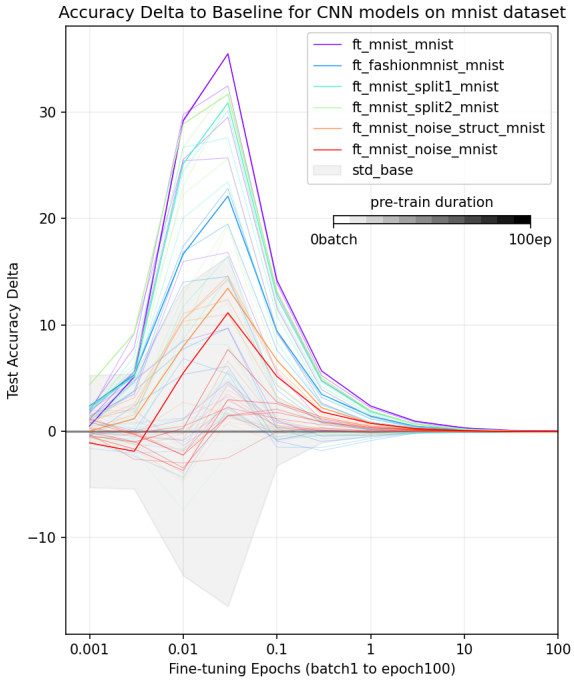
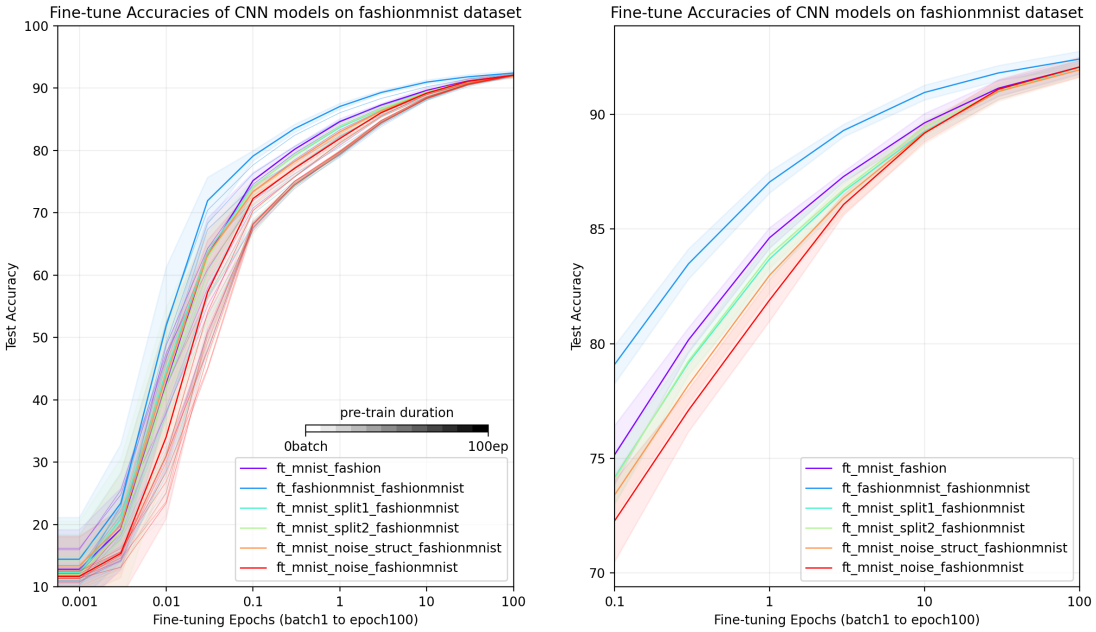


Figure 3.6: MNIST fine-tuning Accuracy delta to baseline

For FashionMNIST as target we get similar results and graphs in Fig. 3.7 as for the previous case, but since FashionMNIST constitutes a harder target problem the archived accuracy values flatten out earlier. As expected FashionMNIST poses the best pre-trainer for itself, therefore we see it distinctively on top in blue.

Again the influence of the pre-training decreases with fine-tuning duration, but the results for the transferred FashionMNIST pre-trained initialization keep a clearly improved and statistically significant performance for up to 30 epochs of fine-tuning.

Especially when looking at the Figure 3.7 (b), that shows the fine-tune test accuracy results for the models pre-trained for 100 epochs for the later fine-tuning checkpoints starting from 100 batches (or 0.1 epoch) to 100 epochs, the improved performance of the FashionMNIST transferred model weights is clear. So much so, that in the case of one epoch it has a significant lead over the noise one of 5%, it takes the red noise model roughly 8 epochs to get to the same accuracy score. If running through the whole target dataset once (1 epoch) would be associated with high costs in regard to time or computation, the differences of 8 times longer training duration to archive the same fine-tune accuracy on the target task would be substantial and could be avoided by just changing the selected pre-train task (both are pre-trained with the same hyperparameters).



(a) FashionMNIST fine-tuning results for all checkpoints

(b) Detail view of later results for 100 epochs pre-trained models

Figure 3.7: Fine-tuning results for CNN models on FashionMNIST

As in the analysis of the MNIST case, to get a clear look at the influence of pre-training on the transfer performance on the FashionMNIST target task the performance delta of all models from Fig. 3.7 (a) and the baseline case of training FashionMNIST from scratch without transferring weights is calculated and displayed in Figure 3.8.

Again the models pre-trained on the noise dataset and most of the earlier structured noise models fall within the non-significant deviation of the baseline case. The earlier batches show the greatest influence of pre-training on improved transfer performance, as is expected since the initialization does not start randomly but with learned representations even if for a different task. In this case, the improvement through transferred initialization decreases with the fine-tune duration as well, but it stays noticeably higher until 100 epochs, showing a significant benefit of transfer learning, especially if considered the limitations and cost of computation and target dataset size.

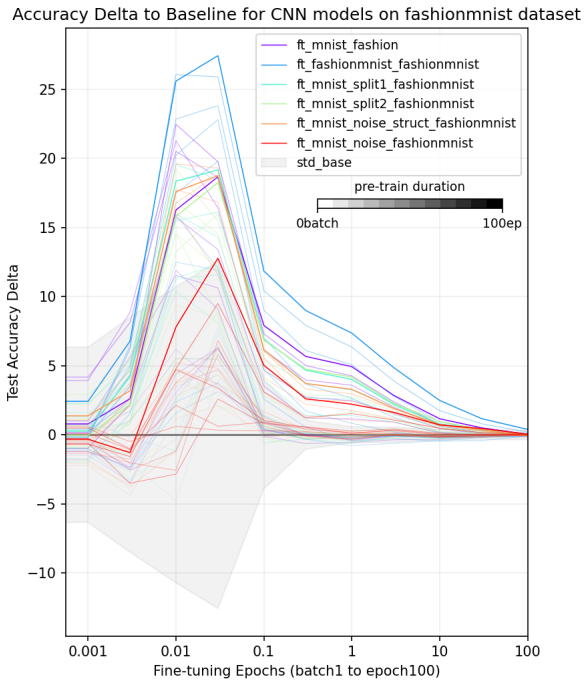


Figure 3.8: FashionMNIST fine-tuning Accuracy delta to baseline

3.2 Results for VGG architecture

3.2.1 Pre-training

Since publicly available pre-trained models were used for the VGG architecture due to computational restrictions (see Section 2.3.1 for details) we can only report on their final performance, since no checkpoints during training are released.

Table 3.1 states the VGG model configuration, the training duration, and test accuracies archived by the 6 pre-trained models gathered. The performance of the classification tasks on the ImageNet, Places365, and Stanford Cars datasets can be measured with two accuracy scores. Commonly in machine learning on complex visual datasets with hundreds of classes with blending boundaries, the top-5 accuracy gets stated, which means that not just the highest probable answer (top-1) but any of the model's 5 highest probability answers matches the label.

Table 3.1: VGG pre-train Accuracy key values

Dataset Name	model config.	training epochs	Top1 Acc.	Top5 Acc.
ImageNet	VGG-16	74	0.7159	0,9038
Places365	VGG-16	90	0.5519	0.8501
Stanford Cars	VGG-16	65	0.8530	0.9590
VGG Face	VGG-16	10	0.9722	
Cifar10	VGG-19	233	0.9243	
CamVid	VGG-16bn	99	0.9857	

Additionally, an untrained random initialized model that functions as a baseline model is used in the evaluation of the metrics and also fine-tuned (see Section 3.2.3). And as a further benchmark for the transfer results we actually train the three target tasks on the VGG architecture according to Section 2.3.1, to be able to identify transfer learning performance improvements (see also Section 3.2.3).

3.2.2 Evaluate metrics

In the following subsections, the results of the three metrics are analyzed for each of the different target tasks on VGG: custom3D, Malaria, and Oxford-Pet.

Following the process outlined in Section 2.3.2 for a fixed subset of each target dataset the activations of every layer get collected from each pre-trained model and additionally from an untrained random initialized VGG-16 model, as a lower benchmark. In the case of the CamVid and Cifar10 datasets the metrics are only obtainable for the feature extractor layers (see Section 2.3.1). As before we focus our attention on the layers that most commonly get transferred, which are 'pool1' to 'pool5' of the convolutional blocks in the feature extractor of the VGG models, but show the results for all layers again in the following.

3.2.2.1 Custom3D

For the visualization of the resulting intrinsic dimension metric on the custom3D target, we find the characteristic 'hunchback' shape described by [Ansuini et al., 2019] for deep neural networks. This refers to the observation that all well-performing models first increase the ID before reducing it till the end of the feature extractor layers. As one can see the untrained random initialized model introduced as the bottom benchmark does not follow the hunchback trend (pink line plot in Figure 3.9). Even though it is the same model architecture, without trained weights it does not transform the inputs in the same way. That there is still a decrease in ID for the random model is due to the nature of convolutional neural networks again, since the parameter amount of each layer reduces with depth.

At the feature extractor output layer 'pool5' the ID values are quite close to each other, except for Cifar10, but taking into account the favorable 'hunchback' shape of [Ansuini et al., 2019] the pre-trained models of ImageNet, Places365, and Stanford Cars appear to create the best intrinsic dimension metric for inputs of the custom3D target.

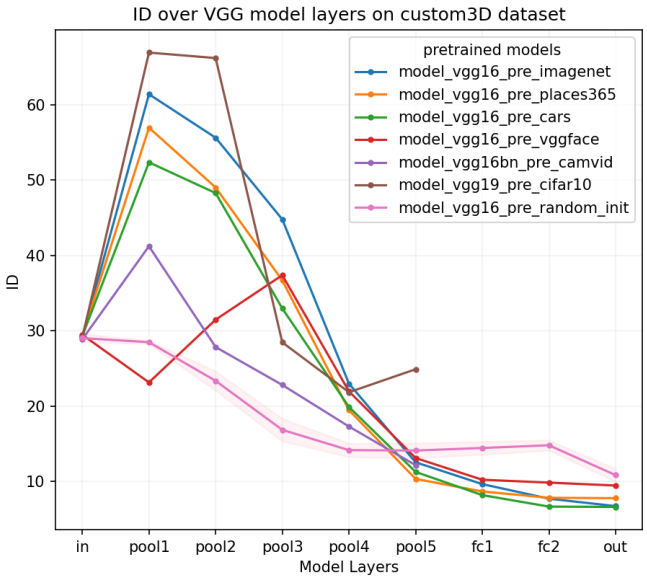


Figure 3.9: ID metric results of VGG models for custom3D

Looking at the cluster differences of the SS metric in Figure 3.10 it shows that for the custom3D target differences become more evident following the 'pool4' layer and on 'pool5' again ImageNet, Stanford Cars, and Places365 perform best. With even more pronounced distinction in the following fully connected layers. It is also clearly visible that the random model is not capable of clustering on the target task and its value and shaded confidence interval stays far apart from the other models.

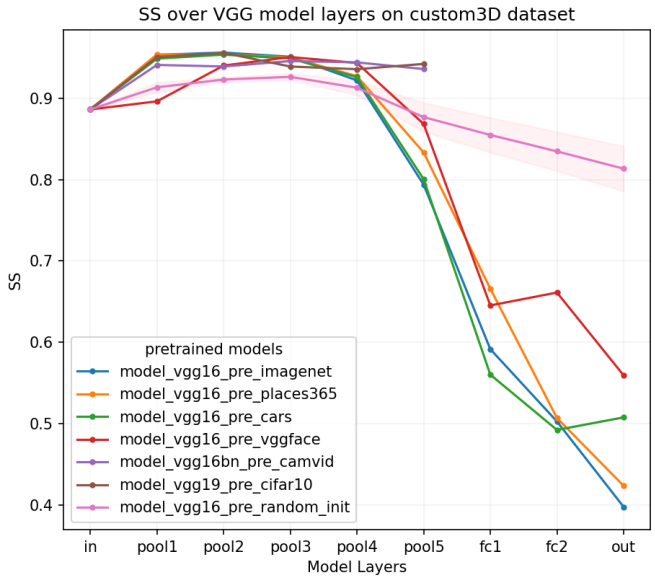


Figure 3.10: SS metric results of VGG models for custom3D

Regrading the RSA metric a clear separation emerges from layer 'pool4' onwards in Fig. 3.11, with ImageNet continuing to outperform the other models significantly on 'pool5' and the classifier layers. The pre-trained models on Places365, Stanford Cars, and in this case VGG Face as well come in second place for RSA metric performance. While the models trained on CamVid and Cifar10 struggle and the random initialized one stays flat throughout.

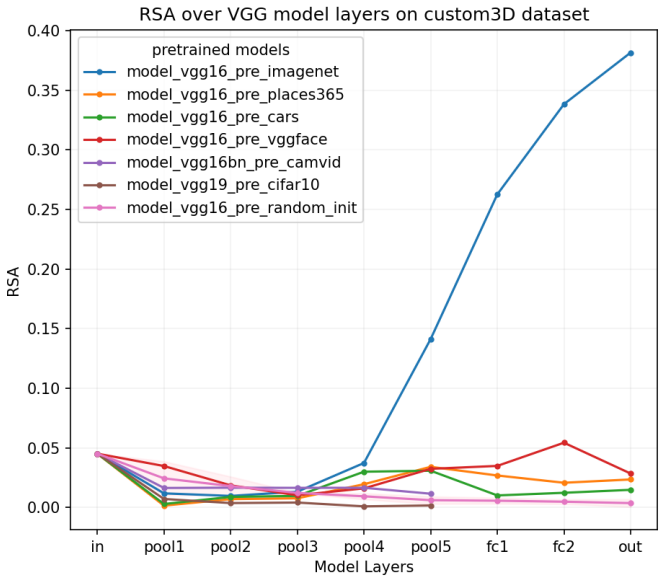


Figure 3.11: RSA metric results of VGG models for custom3D

For all three metrics, the CamVid and Cifar10 pre-trained models appear to perform worst alongside the random initialized one. This could be due to CamVid being semantically quite different to the target task and Cifar10 featuring very small input images, resulting in a difference in architecture and learned representations that affect the metrics on custom3D inputs substantially.

3.2.2.2 Malaria

Extracting the activations for samples of the Malaria target dataset on each pre-trained VGG model and calculating the metrics for them, yield less controlled and uniform results then compared to custom3D, with an order of magnitude difference in scales. This could be due to the high difference in visual stimuli of the blood cell target images compared to the object or scenery-based pre-train datasets. As well as the fact, that the Malaria dataset only contains two classes for which SS clustering and RSA's correlation differences by labels should be expected to deteriorate.

Looking at Fig. 3.12 an extreme peak of the ID value for the ImageNet case immediately catches the eye but considering the embedding dimensionality of 401,408 at layer 'pool2' and 200,704 on 'pool3' of the VGG architecture the intrinsic dimension is still tremendously reduced. Other than that the 'hunchback' shape is not as pronounced but also apparent for the other models, except for the random untrained benchmark one. For the feature extractor output 'pool5' the ID is greatly reduced in magnitude, to values of 27 for ImageNet and quite stagnant 11 for the random model, but maintained separation.

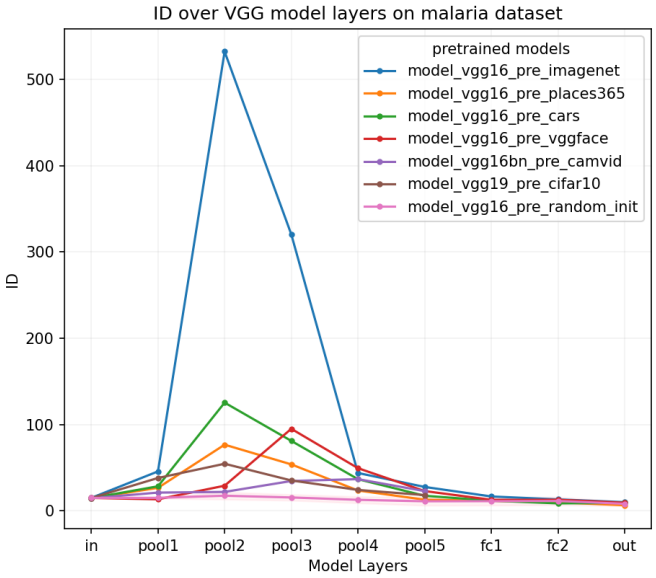


Figure 3.12: ID metric results of VGG models for Malaria

The visualization of the SS metric for the Malaria samples in Fig. 3.13 notably spans a

much smaller range in metric values than in the previous case for custom3D. Therefore the shaded 95% interval of the randomly initialized model appears much wider and some model values can not be significantly separated from the untrained benchmark. The model pre-trained on Stanford Cars performs the only significant cluster separation later on but again, since Malaria only has two classes and all images are of similar blood cells there are viewer discriminating visual features, leading also to looser cluster boundaries and worse performance of the SS metric consequently.

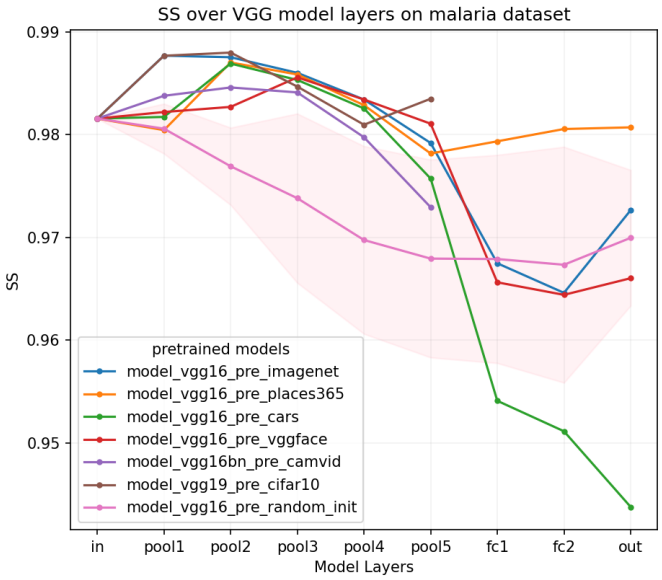


Figure 3.13: SS metric results of VGG models for Malaria

For the RSA metric, the range of results is very small with all values between 0.001 and 0.027. But it shows a clear separation of the pre-trained models on layer 'pool5' and throughout, as well as quite controlled deviation on the random benchmark. The models for CamVid, ImageNet, and VGG Face perform best for Malaria as target and of the others, only Places365 shows significantly improved values over the untrained benchmark.

Intuitively the task of CamVid and Malaria could be more related since on CamVid street scenes get segmented and for example, another car gets identified within a complex scene, while on Malaria the challenge is to identify darker parasited parts of blood cells.

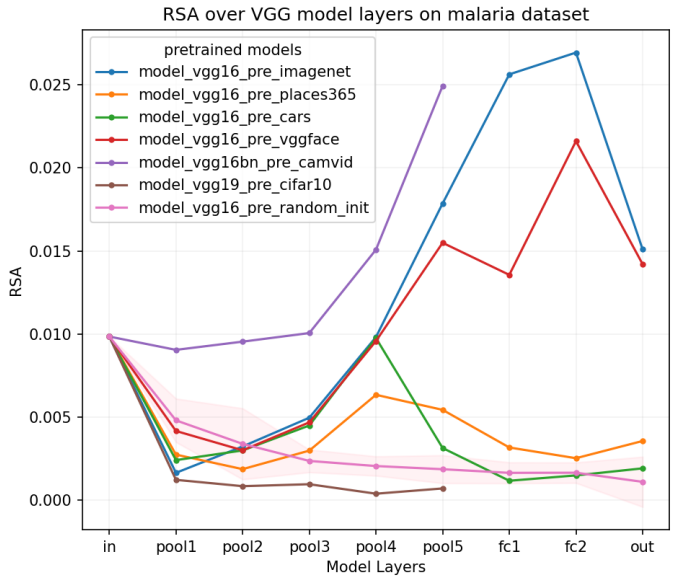


Figure 3.14: RSA metric results of VGG models for Malaria

3.2.2.3 Oxford Pet

For the Oxford Pet dataset as target, we return to mostly the same trends visible for the three metrics as in the custom3D case above. ID shows the sharp incline in earlier layers followed by a monotonically decrease till the output known from the custom3D analysis and [Ansuini et al., 2019].

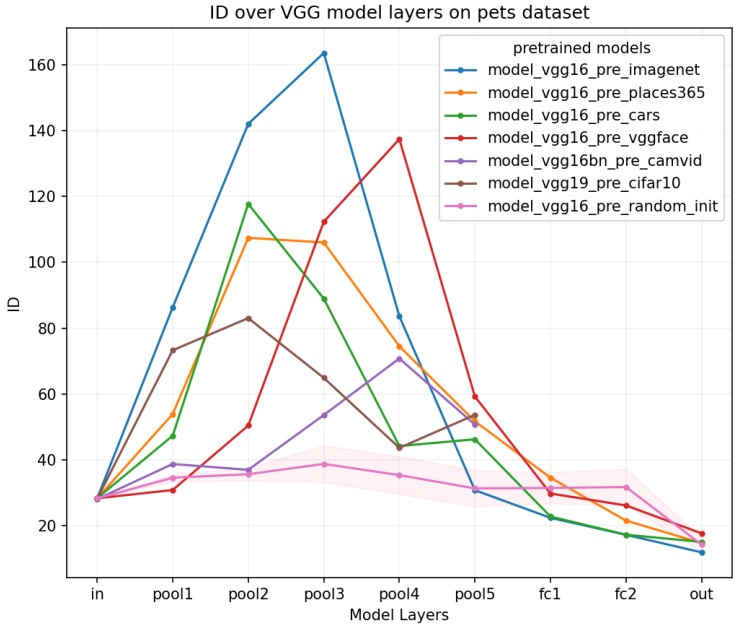


Figure 3.15: ID metric results of VGG models for Oxford Pet

With ImageNet performing best, for highest peak as well as lowest value on 'pool5' and throughout the following layers, followed by Stanford Cars. While the CamVid and Cifar10 pre-trained model also follow the trend they are less pronounced than Places365, which should show better fine-tuning performance even with roughly the same metric score than the other two at layer 'pool5'. As a control benchmark, the untrained random initialized model stays pretty much flat, with controlled variance plotted in shade around it, attesting significance to the other values.

For the sum of squares metric on the Pet target dataset, only ImageNet and Stanford Cars show strong performance, especially on the layer of most interest 'pool5' the other pre-trained models perform worse and lay within the confidence interval of the untrained random model. In total the plotline trends of Fig. 3.17 and particularly the scale are much more similar to the case for the custom3D dataset as target, spanning values from 0.96 to 0.34, instead of the marginal range in the Malaria case.

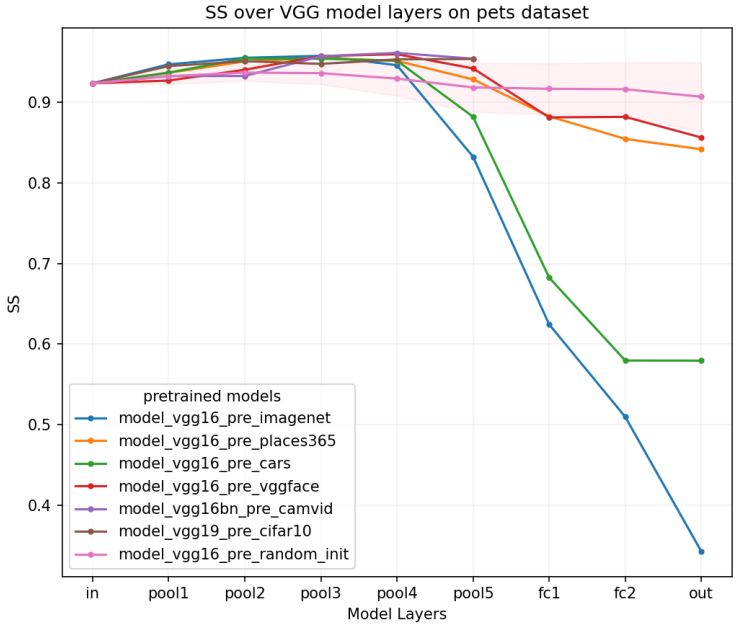


Figure 3.16: SS metric results of VGG models for Oxford Pet

The visualization of the RSA metric on the Oxford Pet dataset as target in Fig. 3.17 reveals a clear separation in the metric outcomes on the last layer of the feature extractor 'pool5'. With ImageNet and Stanford Cars again showing the strongest performance and continue to do so for the following layers. The models pre-trained on VGG Face and Places365 come in tied third but still apart from the others at the very bottom.

Since the ImageNet dataset contains classes of different dog breeds as well, it was expected that it should outperform the other source tasks on the Oxford Pet dataset as target and it very clearly does so for all three metrics.

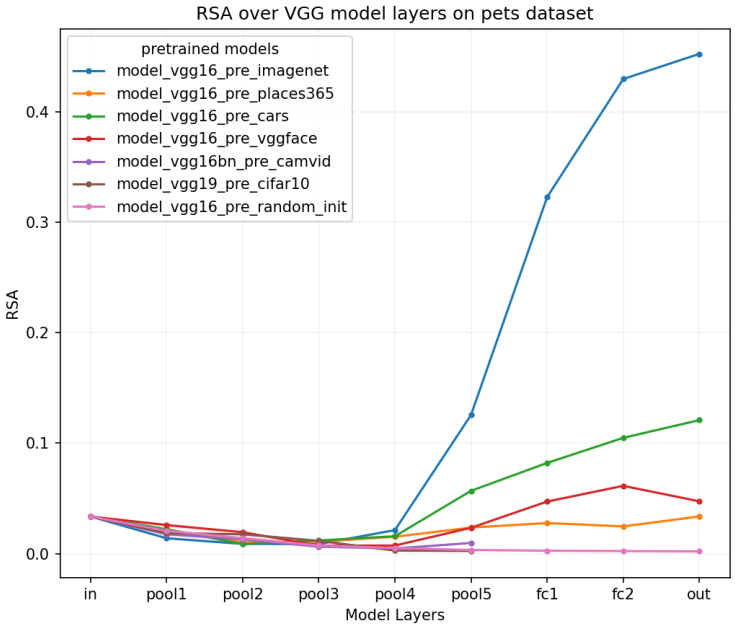


Figure 3.17: RSA metric results of VGG models for Oxford Pet

3.2.3 Transfer Learning outcomes

To be able to evaluate the metrics obtained, the actual transfer performance, meaning the accuracy the models are capable of archiving on the target task after transferring the pre-trained weights and fine-tune them on target, needs to be calculated. The resulting accuracy values of the target models act as the ground-truth for the comparison to the metrics in evaluating their predictive quality in the next Section 3.3.

The following plots show the outcome of transferring and fine-tuning all the pre-trained models with VGG architecture on the three target datasets. They are also extended by two benchmark cases: the first shows performance if the same rules for fine-tuning of Section 2.3.3 are applied to an untrained random initialized model, which means only the last layers get trained. Intuitively that does not suffice to learn a complex task such as classifying custom3D on a deep network like VGG, but it is supposed to only highlight the actually improved performance due to weight transfer, as well as providing a benchmark with the same amount of model weights trained. Compared to the second one, for which a VGG model is trained on custom3D data from scratch, with the process outlined in Section 2.3.1 (see baselines), as if it were pre-training.

custom3D

Fig. 3.18 shows the fine-tuning results of the pre-trained VGG models and the two added benchmarks on the custom3D dataset. Starting from epoch 1 it is clearly visible that the models pre-trained on ImageNet, Stanford Cars and Places365 outperform the others and

archive solid accuracy scores between 80 and 90%. While VGG Face and CamVid are just slightly avoiding being within the 95% interval of the benchmark that is fully pre-trained on custom3D (grey), indicating if the transferred models are significantly different from the just plainly on target trained benchmark case.

As expected the random initialized model does not pick up on any learning within 100 epochs and neither does the fine-tuned Cifar10 model. The latter probably suffers most from the low-resolution input in pre-training, limiting the representational learning performance a VGG network would be capable of, especially since the input changes so much in dimension for the custom3D samples, that comparatively speaking it is as little beneficial to the learning task in target domain as random initialization.

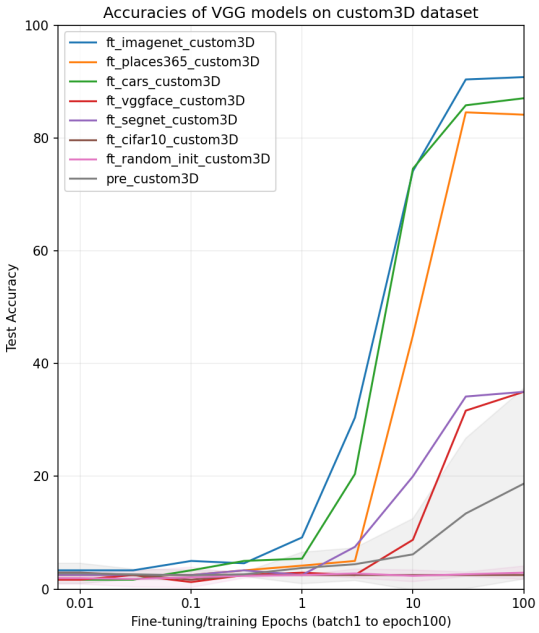
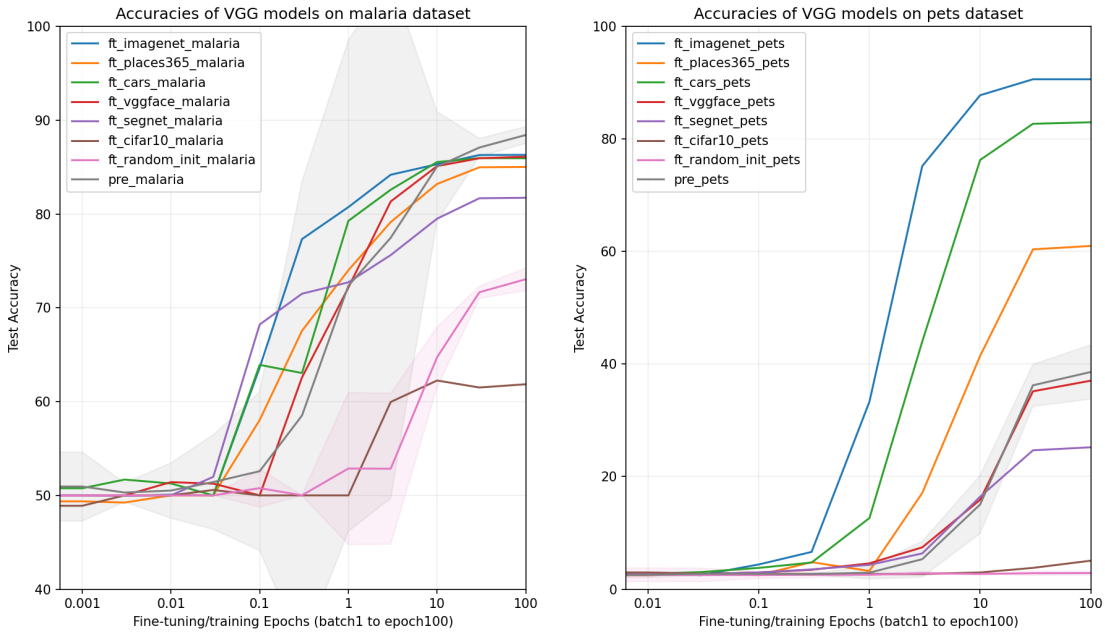


Figure 3.18: Accuracy of VGG models fine-tuned on custom3D

Malaria

Taking all the pre-trained models, transferring their weights, adding the new layers, fine-tune the last convolutional layer as well as the classifier on the Malaria datasets, also do so for a randomly initialized model and fully pre-train a VGG model for Malaria from scratch as benchmark, yields the results visualized in Figure 3.19 (a). Since the dataset only contains two classes the accuracy values, of course, start at 50% already.

Again Cifar10 and the untrained random model perform visibly worst, but in this case, the other models struggle to differentiate from the 'pre' model that is trained from scratch. Revealing that the weight transfer from pre-trained models does provide representations that improve accuracy on the new target task only early on in fine-tune. Visible in the plot e.g. at 0.3 epoch training duration, where the initialization pre-trained on ImageNet shows



(a) ..Malaria **(b)** ..Oxford Pet

Figure 3.19: Fine-tuning results for pre-trained VGG models on..

a roughly 20% performance boost compared to the 'pre-malaria' case. But after 10 epochs that early advantage is mostly compensated and the 'pre' benchmark model not only shows the highest accuracy after 100 epochs but also reduces its 95% confidence interval (of multiple models with different seeds) to do so with significance.

Oxford Pet

The results for accuracy on the Oxford Pet classification task have a higher similarity to the custom3D case again, like the results of the metrics in the previous section. The visualization of the fine-tune results in Fig. 3.19 (b) shows a very clear separation between the results for different pre-trained models, ranging almost the whole scale from 0 to 100. ImageNet and Stanford Cars perform best as pre-trainers to transfer on the target task, with the model pre-trained on ImageNet even archiving 90% accuracy on Oxford Pet. The benchmark VGG-16 model pre-trained from scratch picks up in accuracy later in the training and archives mid-range performance, like the VGG Face and CamVid pre-trained models. Again Cifar10 struggles on the target task and stays mostly at the very bottom with the untrained random model, with no real improvement in accuracy performance during 100 epochs of fine-tuning. Subsequently, only pre-training on ImageNet, Stanford Cars, and Places365 yield better results on the target task than training it from scratch, but they additionally do so much earlier and archive much higher accuracy levels.

Delta

To highlight the accuracies archived after fine-tuning Fig. 3.20 shows the accuracy delta of each model's result to the baseline case of the untrained random model used for fine-tuning. By subtracting the accuracy performance of the baseline from each model's value at every checkpoint. The baseline is plotted in grey and so is its 95% interval as a shade again. Especially in the center plot of the three subplots, shown in the Malaria target task case, the pre-trained models are not able to archive fine-tune accuracies that outperform a random initialized benchmark.

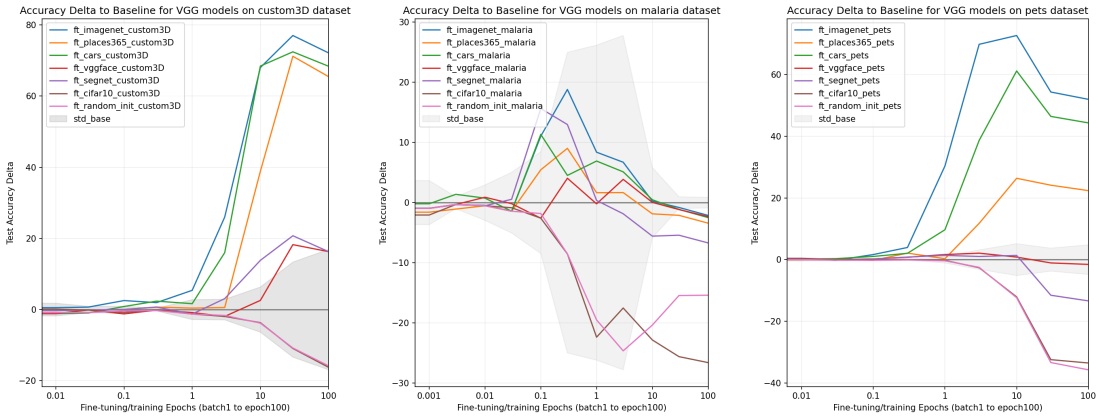


Figure 3.20: Accuracy delta to baseline for all three target tasks on VGG

3.3 Evaluate Transferability Metrics

To quantify the results obtained so far (Sections 3.1 and 3.2) a comparison of the metric outcomes and the fine-tune accuracy archived when actually transferred is needed to provide a statement about the quality of the metrics as a measure of transferability. This concluding analysis of our results is conducted by utilizing Spearman's correlation to rank the results for multiple models and checkpoints obtained on each of the metrics and their actual accuracy score on the target task after fine-tuning and calculate the correlation between these rankings.

Spearman's correlation, also called rank correlation, is equal to the Pearson correlation between the rank values of two variables, therefore providing a better comparative analysis between variables of different scales by only taking the rank and not value magnitude into account. It assesses monotonic relationships, not just linear ones like Pearson correlation. The score for perfect correlation of $+1$ or -1 indicates that each of the variables is a perfect monotone function of the other. Correlation between two variables are high when observations have a similar rank between the two variables ($+1$ for identical ranking), while low when observations have a dissimilar rank between the two variables (-1 for fully opposed ranking).

To check the results gained from rank correlation it is important to understand significance testing for Spearman's correlation. One approach to determining significance of an observed correlation value, meaning to test if it is significantly different from zero, is to use permutation testing. Calculating the probability, given the null hypothesis, for each matching of ranks. But as this is only computationally feasible for a small number of variables to rank, it is not possible to apply it in every one of our cases. Therefore a different approach is needed and found in the approximation using the Fisher transformation. As it is very close to the test values obtained with permutation testing for a small number of variables we use it throughout the following result analysis.

As described in [Bonett and Wright, 2000] the approach parallels the case for the Pearson product-moment correlation coefficient and can be used for confidence intervals and hypothesis tests of Spearman's correlation using the Fisher transformation as well. Basically, by using the transformation to get a sampling distribution of the estimate that is approximately normal, combined with the estimation for the 95% confidence interval using the transformed estimate and adding and subtracting 1.96 times its approximated standard error. For details of the calculation and implementation see Appendix Listing A.4 for the code used to obtain the 95% intervals visible in the following graphs.

To better investigate the predictive quality for transferability of the three metrics and to reduce the amount of data obtained by the experiments we apply some computations beforehand. Especially in the CNN case, since every experiment is conducted for 10 different random initialized models based on different seeds, therefore the results contain for each of the 6 source datasets 10 different models, with 12 checkpoints and the three

metrics based on the extraction of each of the 6 layers (omitting input layer) of every model on each of the 2 target tasks - resulting in a total of 25,920 metric values for the CNN architecture experiments.

So first of all, we take the mean of all seeds again for each model metric, since they were introduced to balance the influence of the starting random initialization and do so by averaging. Resulting in 2,592 data points and since we going to plot three separate plots for each metric and display all 6 layers in it (except for the input layer, since it's the same for each model) and do so for each of the two targets, the total count of data input for each point within these plots is reduced to 72.

In the case of the CNN experiments we transfer and fine-tune each checkpoint of each pre-trained model, and again take the accuracy value on target for 12 checkpoints, doing so for each model of the 6 pre-training datasets and on the two target tasks, results in a dataframe with a total of 17,280 fine-tuning accuracy results. Again the mean over the 10 seeds is taken, leaving us with 864 accuracy results for each of the two targets. One could just take the highest accuracy scores after 100 epochs of fine-tuning as the final results and compare the metrics to just this result. But with that, the significance of boosting the performance on target early in training due to pre-training is lost, since by the time of fine-tuning for 100 epochs the influence of the transferred initialization is often only marginal. Therefore, to shrink all the results of 12 checkpoints down into one value per model expressing the fine-tuning performance, the area under the curve (AUC) of the accuracy graph over the fine-tuning duration is used. And actually, the logarithmic area under the graph (log-AUC) will be utilized in the following.

Since all the plots before (e.g. Fig. 3.5 or Fig. 3.7) are visualized with logarithmic scale it can deceive the perception of the obtained results, if plotted with linear x-axis the earlier results would hardly be visible and the three checkpoint values at 10, 30 and 100 epochs would dominate 90% of the graph. Therefore if the area under the curve is taken on linear scale the early differences have little to no influence and since all fine-tuning results are converging in the case of the simple CNN tasks, they would turn out only marginally different. Taking the area under the curve with a logarithmic scale, like in all the visualizations, we assign equal weights to all logarithmic checkpoints and weight early influences higher than just final output.

Applying the logarithmic area under the curve to each of the 12 checkpoint fine-tuning results only one accuracy value is remaining per model, therefore a total of 72 models per target on CNN. In the following the different cases to inspect the rank correlation for the CNN experiments will be discussed, followed by the VGG result comparison.

In any case, since archiving low intrinsic dimension in later layers is preferred the ranking is inverse to, ideally high, accuracy performance values reached with a model with small ID. Consequently, the rank correlation between ID and fine-tune accuracy should show high negative correlation coefficients on the range from $+1$ to -1. The same is true for the SS metric since the best case is very small sum of squares within classes and large total sum of

squares, meaning minimizing the counter and maximizing the denominator of the metrics fraction. On the contrary, for the RSA metric, a larger distance between the mean of the label diagonal and the off-diagonal values should ideally mean better transfer performance, therefore Spearman's correlation coefficient is expected to be positive.

CNN

For all the result data gathered multiple different analyses are possible. For example, in Figures 3.21 and 3.24 the quality of the three metrics is displayed in a macro way by using all 72 model metric results and log-AUC accuracy values in the rank correlation process. Therefore resulting in only one value per metric and layer, that tells us how well the metric compares across all 6 different pre-training datasets as well as all checkpoints. And it does so statistically significant for all three metrics and in the case of RSA (Fig. 3.21 third subplot) even with a very high correlation coefficient of almost 0.9 on the 'pool2' layer.

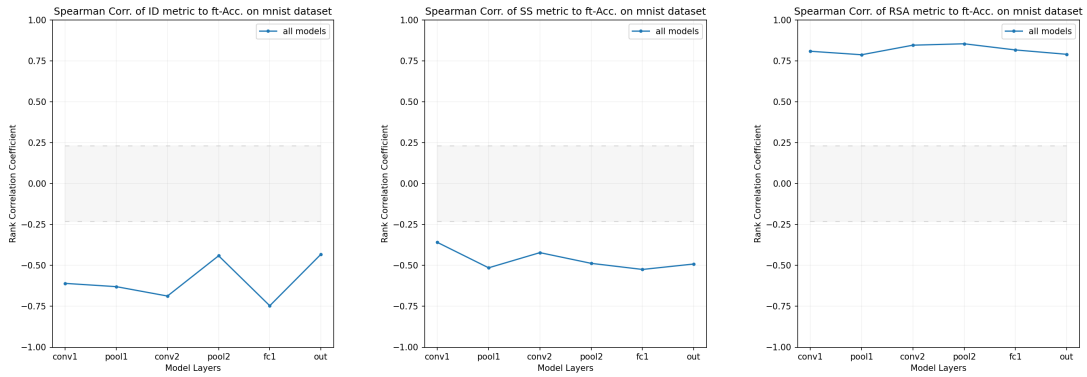


Figure 3.21: Rank Correlation of metrics and accuracy of all MNIST models

While in Figure 3.22 the 72 results are split for each of the 6 pre-train datasets rank correlating the 12 results for each of them with the 12 log-AUC accuracy values obtained. Therefore it shows how well each of the metrics of the models of one dataset correlates with the transfer performance outcomes.

As hoped for, the results of pre-training on the noise dataset show very low and stagnant correlation over the whole model depth (and the fine-tune duration, since the 12 checkpoints are ranked within one line) and do so consistently for each of the three metrics in Fig. 3.22. The structured noise dataset shows the biggest variance in its correlation results, while MNIST and its two splits clearly outperform the rest, show clear trends, and archive very high correlation values. Expect for on ID they also consistently are significantly different from zero by laying outside of the grey interval.

Finally, the third way of analyzing the rank correlation of the results is to do so for each checkpoint, to highlight the influence of the pre-training duration on the correlation. Thus

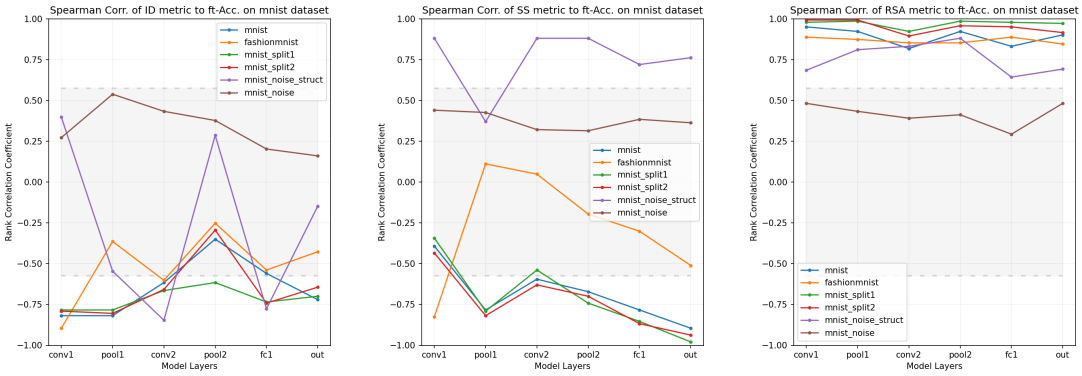


Figure 3.22: Rank Correlation of MNIST models of each pre-train dataset

we split the 72 data points for each layer into 12 for each of the checkpoints from 0 to 100 epochs, resulting in 6 values being ranked and correlated in each case. From that, we derive Figure 3.23, with the early batch checkpoints in lighter color, that show a lot more variance, while the later checkpoints follow the same clear trends. Even though for all three metrics the correlation scores reach high values, unfortunately only the SS metric also archives statistical significance on the 'pool2' layer, but in all cases, the models trained the longest get fairly close to the boundary of the confidence interval. Showing a clear beneficial effect of longer pre-training (over one epoch) on the derived metrics capabilities to predict transferability performance ranks.

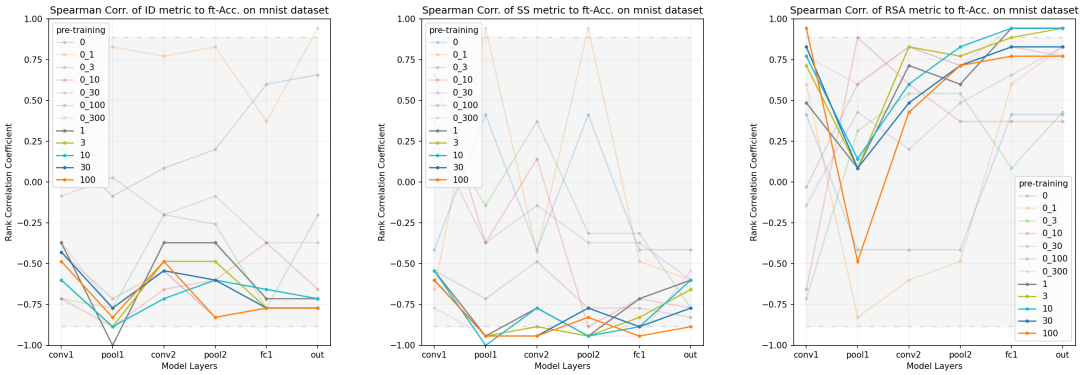


Figure 3.23: Rank Correlation of MNIST models for each pre-train checkpoint

For FashionMNIST as target task the same cornucopia of data is available, therefore a look at all three graphs can be of interest.

First, Figure 3.24 again bundles all results into one ranking for correlation and clearly shows that each of the metrics shows statistical significance on each layer between the metrics on the 6 pre-trained models and the FashionMNIST target performance.

For the comparison using Spearman's correlation on the data pooled by pre-train datasets in Fig. 3.25 the noise and structured noise pre-trained model results start worst for all

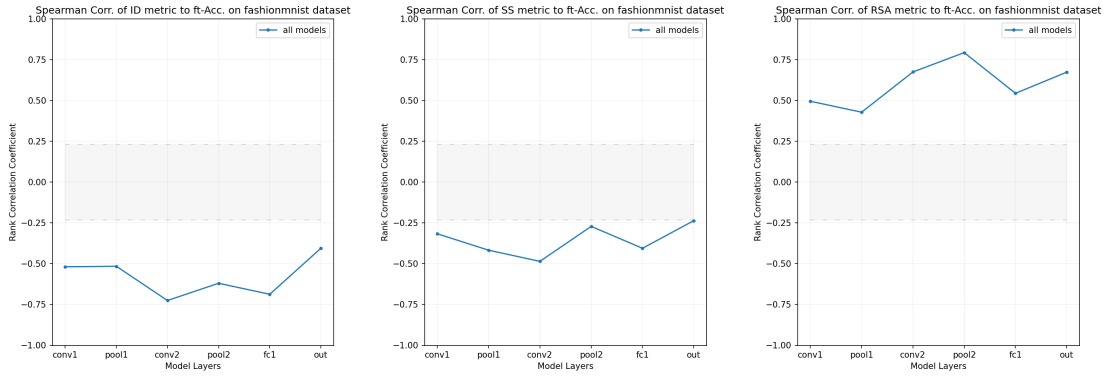


Figure 3.24: Rank Correlation of metrics and accuracy of all FashionMNIST models

three metrics but are not always as clearly separated as in the previous MNIST case. Surprisingly the results for the model checkpoints pre-trained on FashionMNIST, so on the same dataset as the activations are extracted for the metrics and the models get fine-tuned on, do not show the strongest correlation, as one might have suspected. Overall, only the models trained on MNIST and its two splits show a strong and consistent statistically significant correlation on the 'pool2' layer for ID and RSA.

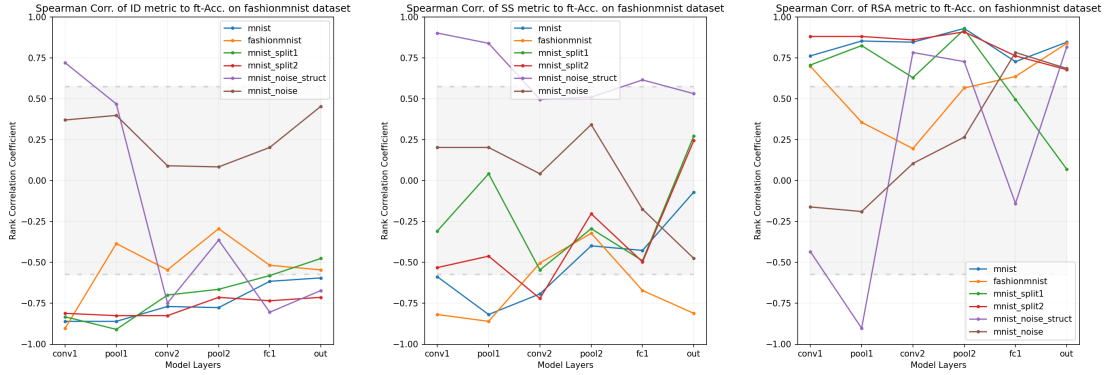


Figure 3.25: Rank Correlation of FashionMNIST models of each pre-train dataset

Figure 3.26 shows the Spearman correlation in regards to the 12 checkpoints for each of the metrics on FashionMNIST as target. While the same trends are visible for the checkpoints later into the pre-training duration, they come with greater variance compared to the case on MNIST and they do not approach the border to statistical significance consistently. Nevertheless, they show improvement along the training duration and reach correlation coefficients bigger than 0.75.

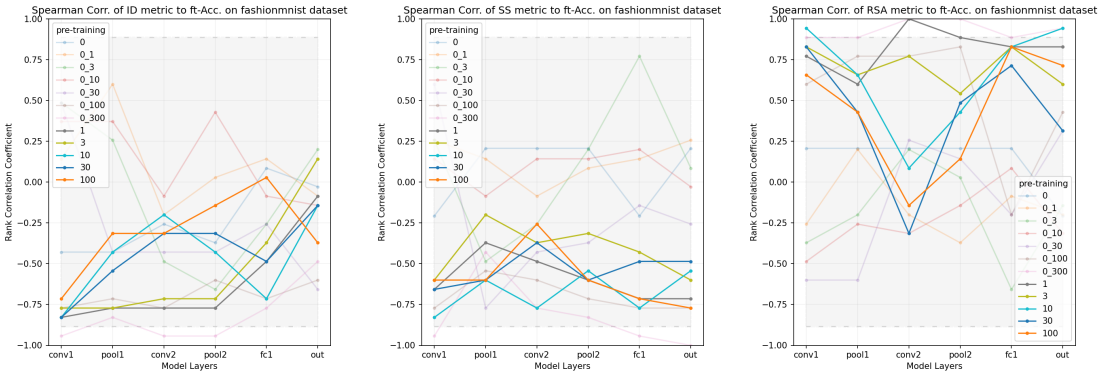


Figure 3.26: Rank Correlation of FashionMNIST models for each pre-train checkpoint

VGG

For the VGG experiments, only one pre-trained model for each source task is used, therefore the three metrics are extracted on 6 models for three targets on 5 layers ('pool1' to 'pool5', discarding input as it is the same for each model and the fully connected layers as they are not available for each model) and again the accuracy values are saved for each of the 12 fine-tuning checkpoints. Additionally, the random initialized model is added like before in the VGG results on metrics and fine-tune accuracy to offer a bottom-line (like previously the noise dataset for CNN).

Due to the limitations in models (and checkpoints) on the source task, the analysis of the Spearman rank correlation results between the three transferability metrics and the fine-tuning accuracy in regards to the pre-train checkpoints as well as the pre-training datasets is not available, since no ranking is possible with only one entry. Nevertheless, bundling all results and apply Spearman's correlation with using the logarithmic area under the curve for combining the 12 fine-tuning results into one result again, is the first way of investigating the rank correlation on VGG too.

To get some more insight into how strong the transferability metrics are correlated to the transferability performance, we split the latter for each of the fine-tuning checkpoints and take the Spearman correlation of each with the metrics. Therefore expanding the single line plot of all models into one for each fine-tune checkpoint, highlighting the influence of fine-tuning duration on the correlation coefficient.

Figure 3.27 shows the rank correlation results between the ID, SS, and RSA metric and the fine-tuning performance on the custom3D dataset on each feature extractor pooling layer of the VGG architecture.

The grey area again marks the 95% confidence interval to be statistically significantly different from zero. The obtained results show very strong correlations of more than 0.75 deeper into the model. For each of the three metrics at layer 'pool5', the coefficient

lands outside or just on the border of the confidence interval, therefore reaching statistical significance.

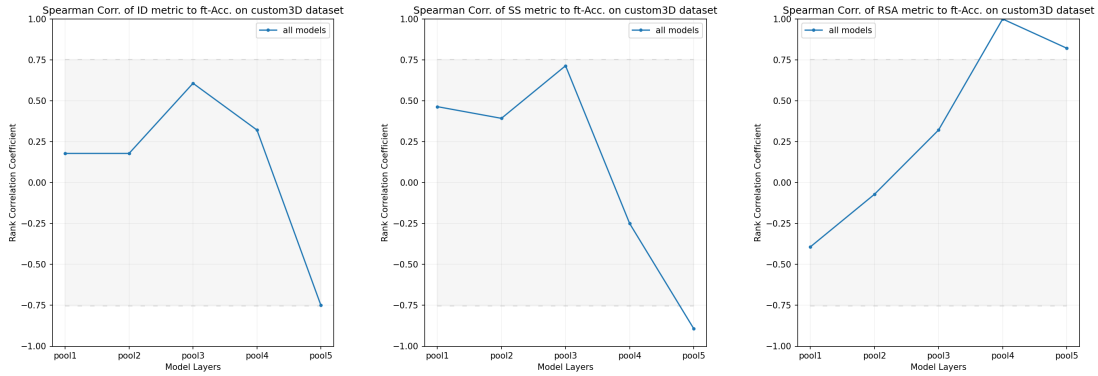


Figure 3.27: Rank Correlation of metrics and accuracy of all models on custom3D

As explained, the other rank correlation plot variations from the CNN experiment results are not possible but additional insight is gained by making the correlation graph of all models more fine-grained in Fig. 3.28 by displaying the results calculated for each fine-tuning checkpoint. Of course, they follow the same trend in later checkpoints as the previous plot, since the more fine-tuned models reaching higher accuracy values contribute a bigger part of the log area under the curve. The correlation results of earlier fine-tune checkpoints, up to one epoch, are shown in lighter colors.

The obtained results show a clear improvement from fine-tuning durations of only a few batches up to the ones trained the longest, which reach a very strong correlation of more than 0.75 in each of the three metrics at layer 'pool5' with statistical significance.

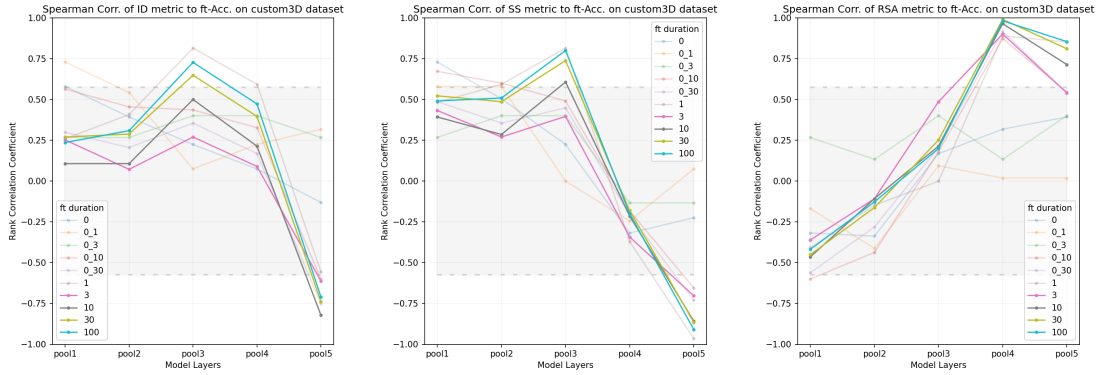


Figure 3.28: Rank Correlation of metrics and accuracy on custom3D for each fine-tuning checkpoint

For the experiments with the Malaria dataset as target the results obtained for the metrics, as well as the fine-tuning performance in the target domain showed less clear trends than for custom3D or Oxford Pet. The rank correlation results continue to do so, with only

RSA showing high values and statistical significance in later layers in Fig. 3.29. While the correlations for ID stay stagnant and SS starts to move towards expected negative correlation but does not reach a significant magnitude.

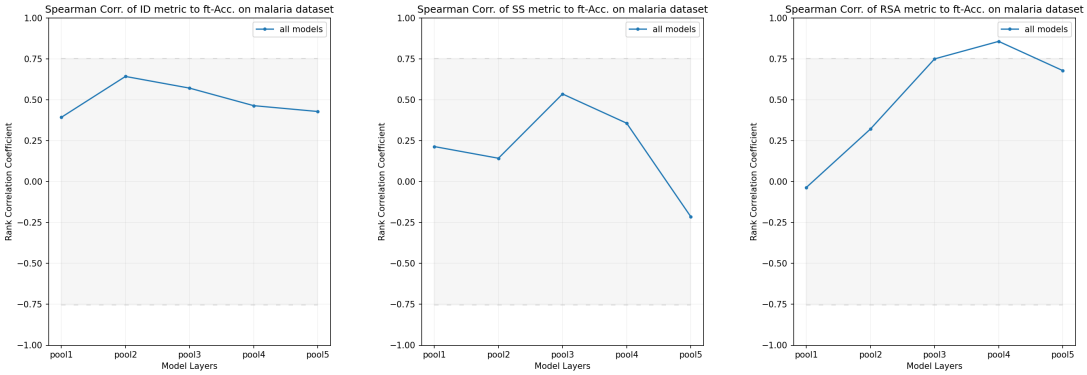


Figure 3.29: Rank Correlation of metrics and accuracy of all models on Malaria

Expanding the plot for each of the fine-tuning checkpoints reveals that on Malaria as target the models fine-tuned for a longer duration archive high positive correlation results on all metrics. But only for RSA that is the actual desired behavior, for ID the intuition is that a reduction to small dimensionality on the end of the convolutional layers benefits the transfer performance and while for SS the correlations at least start to inverse for the last layer on later checkpoints, ID stays stagnant. Since the fine-tuning results for Malaria in Section 3.2.3 already showed that the influence of pre-training is only marginal, it was to be expected that the Spearman correlation would not be conclusive.

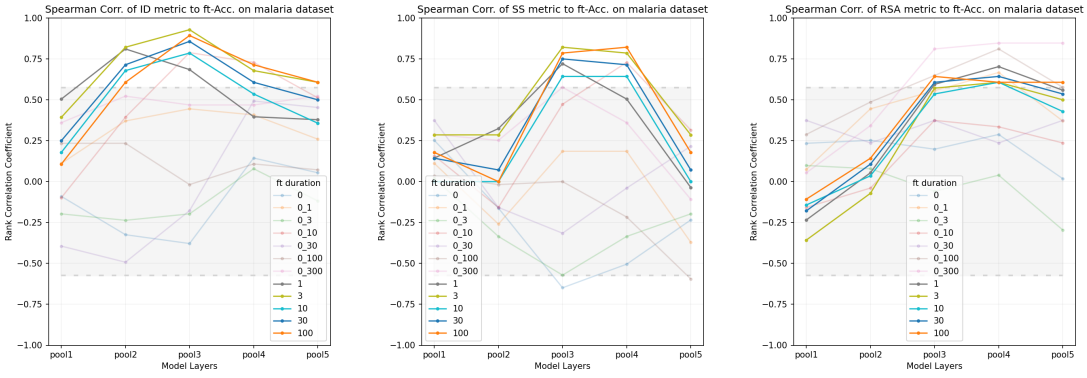


Figure 3.30: Rank Correlation of metrics and accuracy on Malaria for each fine-tuning checkpoint

Finally, Figure 3.31 displays the Spearman correlation results for the third target on the VGG architecture, Oxford Pet, again for each feature extractor pooling layer, with the three plots showing the correlations of the ID, SS, and RSA metrics to fine-tune accuracy respectively.

The trend of the ID and SS plot heading towards negative correlation is more pronounced

again compared to Malaria, but less so than for custom3D, failing to leave the confidence interval. The rank correlation between the RSA metric and fine-tune accuracy on the other hand almost reaches perfect correlation in the last pooling layer.

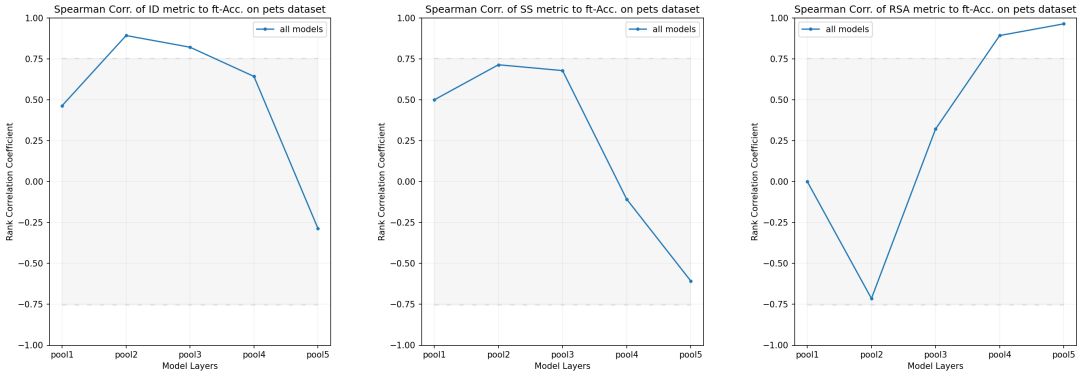


Figure 3.31: Rank Correlation of metrics and accuracy of all models on Oxford Pet

Investigating these results on Oxford Pets further by calculating the rank correlation for each fine-tune checkpoint improves the results by highlighting that for the center plot in Figure 3.32 two of the later model results do archive statistical significant correlations. And while they still do not in case of ID correlation, the trend is more accentuated the more the model has been trained and does not remain stagnant like in the Malaria case.

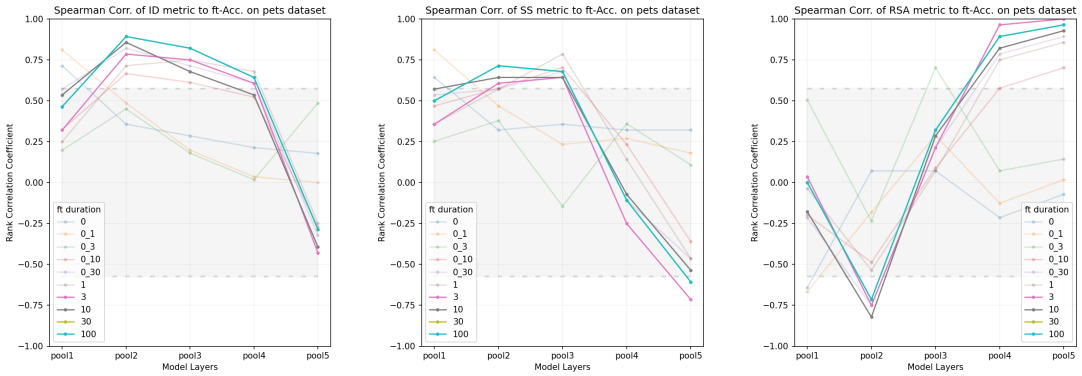


Figure 3.32: Rank Correlation of metrics and accuracy on Oxford Pet for each fine-tuning checkpoint

In summary, we observed high correlation values and statistical significance in most of the evaluations. For cases that yield less convincing results, it is in line with their metrics or fine-tuning outcomes. Therefore we showed significant potential that the three metrics set out with, actually comprise predictive quality for transfer performance.

With all other parameters fixed and a fine-tuning duration of at least 1 epoch, e.g. the RSA metric score can be calculated just using the pre-train models (that ideally are publicly available) and a small subset of the target data (500 to 1,000 samples), which will provide

a ranking of the pre-trained models that strongly coincides with actual fine-tune outcome after transfer, before doing so.

4 Conclusions

Concluding this thesis, we reflect upon the satisfaction of the research objective defined at the beginning and the accomplishment of the initially defined goals.

Furthermore, the major contributions made get reiterated and summarized, before critically reflecting on the aspects that still provide the potential for future work.

4.1 Review of Research Question

This thesis set out to investigate the understanding of transfer learning performance between visual tasks on deep neural networks. Especially, the potential for a transferability metric, capable of guiding the selection of source task and model before transferring by utilizing only the pre-trained model and a small amount of target data. Since the benefit of transfer learning is proven but unknown before actually conducted, a significant practical use case would arise from the capability to predict the best candidate for transfer.

Thereupon three different candidate metrics spanning from recent academic research to established machine learning methods to neuro-science-inspired procedures got introduced. To investigate their potential thoroughly, multiple experiments were set up and written in program code, leveraging different model architecture complexities as well as a diverse set of datasets. Each thoroughly introduced, as well as visualized and discussed for all the obtained results.

Especially the application of Spearman's correlation as the final step of analysis upon the obtained results for metrics and transfer performance archived the goal of examining the fulfillment of the requirement to identify a genuine transferability metric. Since all three metrics showed a high rank correlation in regards to the 'ground-truth' of actually transferring and fine-tuning each model and particularly the metric inspired by representational similarity analysis did so consistently across all model architectures and target domains with statistically significant results.

The experiments were conducted following best-practices while trying to reduce the influence of mere hyperparameters on the result evaluation by keeping many of them fixed over all experiment runs performed. But consequently, this approach also leaves room for improvements to investigate many more setup variants, when not restricted by time and computation capabilities as much as in this thesis.

4.2 Contributions

The main contribution constitutes the introduction, definition, application, and evaluation of metrics previously not or not in this variant applied to answering the question of transferability.

Intrinsic dimension had not been applied to transfer learning previously and representational similarity analysis only very recently and not in the way we simplified the approach regarding the metric derived from the representational dissimilarity matrices. Therefore, novel ideas of very recent research have been incorporated into this approach.

Furthermore, only for the pre-trained model on the VGG architecture external results were adopted, otherwise, all models were trained for this thesis and their weights will be provided to facilitate further research. As well as the implementation of all the experiment processes, analytical computations and resulting visualizing programmed for generating the results of this work.

Following many of the best practices of the machine learning research community, general understanding and higher comparability to existing insights are facilitated. Of course, also the obtained results are improved by measures like generating 10 different seeds for random weight initialization to be able to better cope with the influence of primary model weights on the results and asses them with much greater statistical thoroughness and significance by taking the mean and adding confidence intervals.

The inclusion of a diverse set of visual datasets with a wide range of demanding, complex features and the addition of benchmark cases to most of the evaluations contributes to the wide range of results archived and potential for comparative insight. In regard to the main objective of examining transferability, the results for each of the metrics were calculated not just on the output layer of the transferred part, but on each layer of the models. Facilitating multidimensional insight regarding model depth, source dataset, and pre-training duration. Combined with the results obtained from transferring and fine-tuning all our models, which scored exemplary accuracy performances on all the target tasks, we were able to reach high correlations, within the grasp of results of recent leading research.

In summary, high correlation values and statistical significance can be archived in most of the evaluations between the introduced metrics and the actual transferring 'ground-truth'. Therefore we showed, that the three metrics set out with actually comprise predictive quality for transfer performance. Meaning, when faced with the selection of one of many given pre-trained model as initialization for learning a target task via transfer and additionally at least a small set of labeled data from the desired target domain is available, we can extract the activations of the last feature extractor layer for all pre-trained models, when inputting target samples and calculate the RSA metric. Under the restriction that parameters are set in a similar way to our experiments and that the actual fine-tuning after transfer will at least be conducted for more than just a few batches, the RSA metric will provide a

ranking of the pre-trained models that has the model providing the best possible transfer performance with high probability at the top.

Compared to other measures set to quantify transferability estimation our metrics are easier to obtain and preserve their interpretability. Furthermore, they work under fewer restrictions and archive, at least for our limited experiments, competing correlation scores when compared to transfer performance.

Since this thesis intends to be of an encompassing nature, the introduction into the subject, the study of related work, as well as the thorough definition and explanation of all methods used, also presents a contribution and an open invite to base further research upon it.

4.3 Future Work

As the very active research in the field of quantifying transfer learning and the transfer of representations shows (see. Section 1.2) and many relating academic papers being released during the course of just working on this thesis, there remain many interesting research opportunities and novel ideas to be discovered.

As mentioned previously the experiments were conducted to the best of the capabilities of this thesis, following best-practices and ways of focusing on the results for the transferability metrics and not necessarily investigating all possible options for the setups. While also being restricted in computational resources and time. Therefore the field offers much more empirically explorative possibilities to research. As well as a multitude of possible improvements to the approaches laid out in this work, like adding more model architectures to investigate the quality of the metrics obtained from their activations. Also, the amount of available data is a big restraint in real-world applications, hence varying the number of samples for the pre-training as well as in case of fine-tuning will certainly yield novel insights and possible ways to improve transferability metrics. Furthermore, the investigation of which layers to include in transfer of DNNs promises another big angle for improvements, as it is integral to the transfer process and a lot of research is already particularly focused around it. Generally speaking, the whole hyperparameter space could yield results proving further insight, which was purposely neglected in this study, to focus on just architecture, dataset, and duration.

Transfer learning appears to be most popularly applied and researched for visual tasks, but further work lies ahead in adopting the metrics on completely different task domains like natural language processing too.

Finally, a combination of multiple metrics taking into account different domains like the ones of this thesis do with dimensionality, clustering and correlation could yield even stabler results, if computational demand can be deferred.

A Appendix

Attached in this Appendix some extensions and additions can be found. Mainly Section A.1 provides additional information to the program code created for this thesis, while Section A.2 includes further images and graphs.

A.1 Developed program code

A.1.1 Environments, Frameworks, and Packages

The experiments, analysis, and visualizations are created using the integrated development environment PyCharm for the Python programming language.

For training and manipulating neural networks, the PyTorch ML framework was used and with it the CUDA toolkit for performing the model training on GPU hardware.

At this point, I would like to thank Nvidia's hardware grant program for providing the GPU hardware used on the server cluster of the NI chair at TU Berlin to compute the results in this thesis.

Version control was maintained using Git with a GitHub repository, where the complete program code is publicly released at `https://github.com/flombe/thesis`.

A.1.2 Program Code

For the complete code please refer to the mentioned repository, but parts of it, that were mentioned in the thesis, are stated below.

Listing A.1: MNIST_noise_struct dataset definition

```
"""
Create MNIST_noise_struct dataset
"""

test_data, test_labels = torch.load(join(dataset_dir, 'processed/test.pt'))
test_data = np.random.uniform(0, 1, (10000, 784))

# percentage of values that share the same value per label
ratio = 0.1

for label in range(0, 10):
    # get random pixel indices
```

```
    idx = np.random.choice(784, int(784 * ratio), replace=False)
    # same distribution values assigned to all label values
    label_values = np.random.uniform(0, 1, int(784 * ratio))
```

Listing A.2: Malaria dataset transformations

```
"""
Extract from the Malaria dataset class for defining the dataloaders showing the
    transformations applied to Malaria samples
"""

from torch.utils.data import DataLoader
from torchvision import datasets, transforms

class Malaria(Dataset):
    def __init__(self, dataset_dir, device):
        self.dataset_dir = dataset_dir
        self.loader_args = {'num_workers': 10, 'pin_memory': True} if device.type == 'cuda'
            else {}

        self.train_data = datasets.ImageFolder(os.path.join(dataset_dir, 'train'), self.
            get_train_transform())
        self.test_data = datasets.ImageFolder(os.path.join(dataset_dir, 'test'), self.
            get_test_transform())
        self.class_names = self.train_data.classes

    def get_train_transform(self):
        transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.ColorJitter(0.05),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(20),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ])
        return transform

    def get_test_transform(self):
        transform = transforms.Compose([
            transforms.RandomResizedCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
        ])
        return transform
```

Listing A.3: Extract function on VGG-16

```
"""
Extract function for VGG models to save activations
"""

#     Sequential(
#   (0): Conv2d (3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (1): ReLU(inplace)
#   (2): Conv2d (64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (3): ReLU(inplace)
#   (4): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
#   (5): Conv2d (64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (6): ReLU(inplace)
#   (7): Conv2d (128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (8): ReLU(inplace)
#   (9): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
#   (10): Conv2d (128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (11): ReLU(inplace)
#   (12): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (13): ReLU(inplace)
#   (14): Conv2d (256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (15): ReLU(inplace)
#   (16): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
#   (17): Conv2d (256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (18): ReLU(inplace)
#   (19): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (20): ReLU(inplace)
#   (21): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (22): ReLU(inplace)
#   (23): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
#   (24): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
#   (25): ReLU(inplace)
#   (26): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (27): ReLU(inplace)
#   (28): Conv2d (512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
#   (29): ReLU(inplace)
#   (30): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1))
# )

def extract_all(self, x, verbose=False):
    # from pooling layers 1,2,3,4,5
    out0 = x
    out1 = self.features[4](F.relu(self.features[2](F.relu(self.features[0](x)))))
    out2 = self.features[9](F.relu(self.features[7](F.relu(self.features[5](out1)))))
    out3 = self.features[16](F.relu(self.features[14](F.relu(self.features[12](F.relu(self.
        features[10](out2)))))))
    out4 = self.features[23](F.relu(self.features[21](F.relu(self.features[19](F.relu(self.
        features[17](out3)))))))
    out5 = self.features[30](F.relu(self.features[28](F.relu(self.features[26](F.relu(self.
        features[24](out4)))))))

    # from the classifier part
    out6 = F.relu(self.classifier[0](out5.view(out5.size(0), -1)))
    out7 = F.relu(self.classifier[3](out6))
    out8 = F.relu(self.classifier[6](out7))

    return [out0, out1, out2, out3, out4, out5, out6, out7, out8]
```

Listing A.4: Fisher 95% confidence interval

```
"""
Calculate 95% confidence intervals for the number of the Spearman rank correlation elements
    using Fisher transformation
"""
def fisher_95int(num):
    low = []
    up = []
    r = 0  # null hypothesis
    stderr = 1.0 / math.sqrt(num - 3)
    delta = 1.96 * stderr
    lower = math.tanh(math.atanh(r) - delta)
    upper = math.tanh(math.atanh(r) + delta)
    print("lower %.6f upper %.6f" % (lower, upper))
    low.append(lower)
    up.append(upper)

    return low, up
```

A.2 Additional Figures

Finally, in the following some additional visualizations are provided, enriching and extending parts of the thesis and providing optional insight.

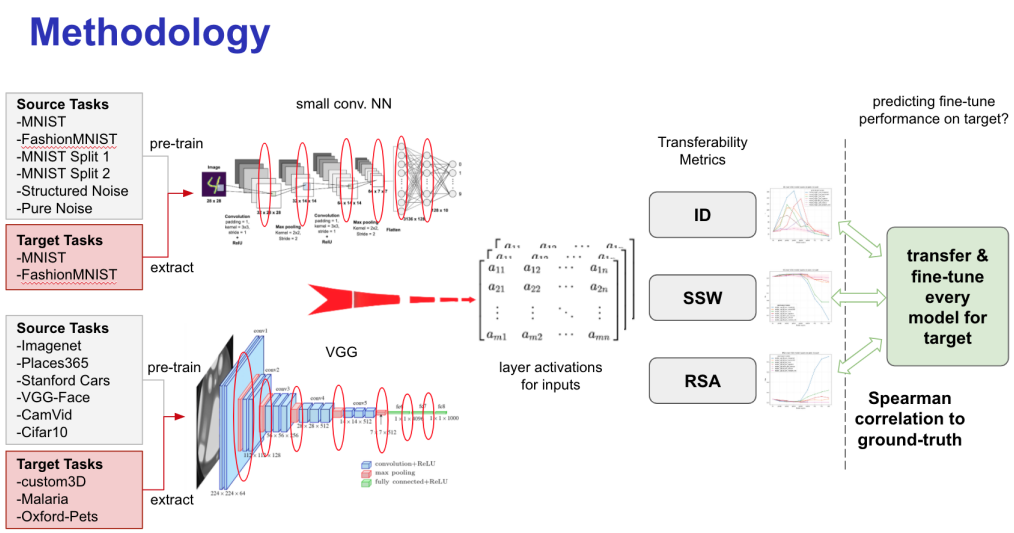


Figure A.1: Experimental methodology scheme, from thesis presentation

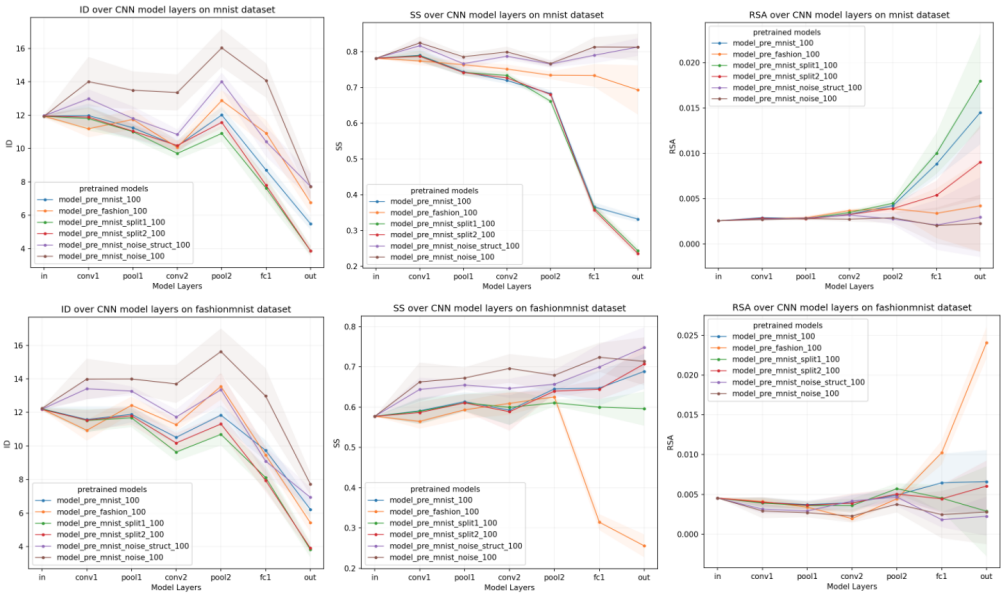


Figure A.2: Comparison overview of all metric plots for both targets on CNN

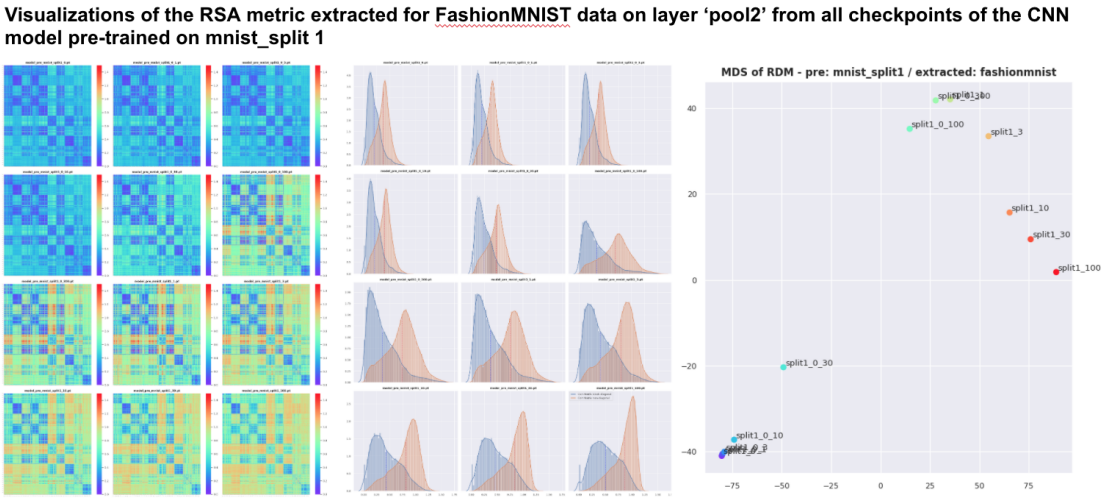


Figure A.3: Additional visualizations applied and tested out for RSA metric, notably multi-dimensional scaling (MDS)

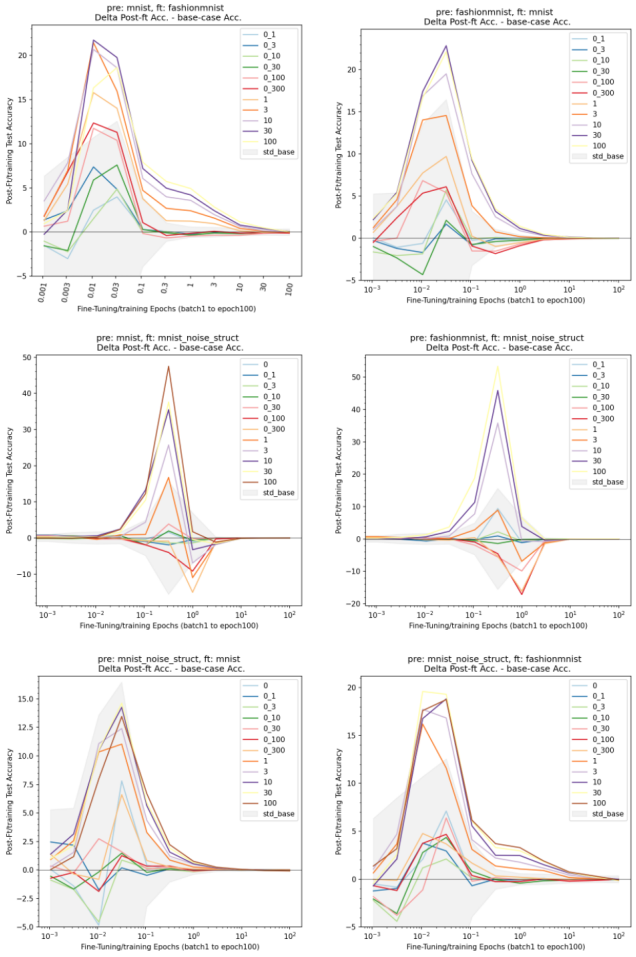


Figure A.4: More detailed looks at accuracy deltas of fine-tuning results to baseline case

List of Figures

2.1 VGG-16 network architecture visualization, from [Ferguson et al., 2017] . . 11
2.2 MNIST dataset samples from [LeCun et al., 1998] 13
2.3 FashionMNIST dataset samples from [Xiao et al., 2017] 14
2.4 Random noise dataset samples 15
2.5 ImageNet dataset samples from [Russakovsky et al., 2015] 16
2.6 Places365 dataset samples from [Zhou et al., 2017] 16
2.7 Stanford Cars dataset samples from [Krause et al., 2013] 17
2.8 VGG Face dataset samples from [Parkhi et al., 2015] 17
2.9 CamVid video frame example and corresponding labeled frame from [Brostow et al., 2009] 18
2.10 Cifar10 dataset classes and samples 19
2.11 Custom3D dataset from [Vascon et al., 2018] 20
2.12 Malaria dataset samples 20
2.13 Oxford Pet dataset samples 21
2.14 TWO-NN estimation procedure: estimating ID from the statistics of nearest neighbour distances, from [Ansuini et al., 2019] 24
2.15 RSA: calculation of RDM and delta of the mean diagonal values 27

3.1 Pre-train Accuracy on CNN models 36
3.2 ID metric results for CNN models 37
3.3 SS metric results for CNN models 38
3.4 RSA metric results for CNN models 38
3.5 Fine-tuning results for CNN models on MNIST 39
3.6 MNIST fine-tuning Accuracy delta to baseline 40
3.7 Fine-tuning results for CNN models on FashionMNIST 41
3.8 FashionMNIST fine-tuning Accuracy delta to baseline 42
3.9 ID metric results of VGG models for custom3D 44
3.10 SS metric results of VGG models for custom3D 45
3.11 RSA metric results of VGG models for custom3D 45
3.12 ID metric results of VGG models for Malaria 46
3.13 SS metric results of VGG models for Malaria 47
3.14 RSA metric results of VGG models for Malaria 48
3.15 ID metric results of VGG models for Oxford Pet 48
3.16 SS metric results of VGG models for Oxford Pet 49
3.17 RSA metric results of VGG models for Oxford Pet 50

3.18 Accuracy of VGG models fine-tuned on custom3D 51
3.19 Fine-tuning results for pre-trained VGG models on.. 52
3.20 Accuracy delta to baseline for all three target tasks on VGG 53
3.21 Rank Correlation of metrics and accuracy of all MNIST models 56
3.22 Rank Correlation of MNIST models of each pre-train dataset 57
3.23 Rank Correlation of MNIST models for each pre-train checkpoint 57
3.24 Rank Correlation of metrics and accuracy of all FashionMNIST models . . 58
3.25 Rank Correlation of FashionMNIST models of each pre-train dataset 58
3.26 Rank Correlation of FashionMNIST models for each pre-train checkpoint . 59
3.27 Rank Correlation of metrics and accuracy of all models on custom3D 60
3.28 Rank Correlation of metrics and accuracy on custom3D for each fine-tuning checkpoint 60
3.29 Rank Correlation of metrics and accuracy of all models on Malaria 61
3.30 Rank Correlation of metrics and accuracy on Malaria for each fine-tuning checkpoint 61
3.31 Rank Correlation of metrics and accuracy of all models on Oxford Pet . . . 62
3.32 Rank Correlation of metrics and accuracy on Oxford Pet for each fine-tuning checkpoint 62

A.1 Experimental methodology scheme, from thesis presentation 72
A.2 Comparison overview of all metric plots for both targets on CNN 72
A.3 Additional visualizations applied and tested out for RSA metric, notably multi-dimensional scaling (MDS) 73
A.4 More detailed looks at accuracy deltas of fine-tuning results to baseline case 73

List of Tables

2.1 Overview and key data of datasets used 12

3.1 VGG pre-train Accuracy key values 43

Bibliography

[Achille et al., 2019] Achille, A., Lam, M., Tewari, R., Ravichandran, A., Maji, S., Fowlkes, C. C., Soatto, S., and Perona, P. (2019). Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6430–6439.

[Ansuini et al., 2019] Ansuini, A., Laio, A., Macke, J. H., and Zoccolan, D. (2019). Intrinsic dimension of data representations in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 6111–6122.

[Badrinarayanan et al., 2017] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[Bennett, 1969] Bennett, R. (1969). The intrinsic dimensionality of signal collections. *IEEE Transactions on Information Theory*, 15(5):517–525.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Bishop et al., 1995] Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

[Bonett and Wright, 2000] Bonett, D. G. and Wright, T. A. (2000). Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28.

[Brostow et al., 2009] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

[Dwivedi et al., 2020] Dwivedi, K., Huang, J., Cichy, R. M., and Roig, G. (2020). Duality diagram similarity: a generic framework for initialization selection in task transfer learning. In *European Conference on Computer Vision*, pages 497–513. Springer.

[Dwivedi and Roig, 2019] Dwivedi, K. and Roig, G. (2019). Representation similarity analysis for efficient task taxonomy & transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12387–12396.

[Facco et al., 2017] Facco, E., dâ€™Errico, M., Rodriguez, A., and Laio, A. (2017). Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific reports*, 7(1):1–8.

[Farabet et al., 2012] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2012). Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929.

[Ferguson et al., 2017] Ferguson, M., Ak, R., Lee, Y.-T. T., and Law, K. H. (2017). Automatic localization of casting defects with convolutional neural networks. In *2017 IEEE international conference on big data (big data)*, pages 1726–1735. IEEE.

[Fisher et al., 1921] Fisher, R. A. et al. (1921). 014: On the" probable error" of a coefficient of correlation deduced from a small sample.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Krause et al., 2013] Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia.

[Kriegeskorte et al., 2008] Kriegeskorte, N., Mur, M., and Bandettini, P. A. (2008). Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:4.

[Krizhevsky et al., 2009] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2:396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Liu et al., 2019] Liu, H., Long, M., Wang, J., and Jordan, M. I. (2019). Towards understanding the transferability of deep representations. *arXiv preprint arXiv:1909.12031*.

[Long et al., 2013] Long, M., Wang, J., Ding, G., Sun, J., and Yu, P. S. (2013). Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207.

[MacQueen et al., 1967] MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

[McClure and Kriegeskorte, 2016] McClure, P. and Kriegeskorte, N. (2016). Representational distance learning for deep neural networks. *Frontiers in computational neuroscience*, 10:131.

[Mehrer et al., 2018] Mehrer, J., Kriegeskorte, N., and Kietzmann, T. (2018). Beware of the beginnings: intermediate and higherlevel representations in deep neural networks are strongly affected by weight initialization. In *Conference on Cognitive Computational Neuroscience*.

[Mehrer et al., 2020] Mehrer, J., Spoerer, C. J., Kriegeskorte, N., and Kietzmann, T. C. (2020). Individual differences among deep neural network models. *Nat Commun 11, 5725*.

[Nguyen et al., 2020] Nguyen, C. V., Hassner, T., Archambeau, C., and Seeger, M. (2020). LEEP: A New Measure to Evaluate Transferability of Learned Representations.

[Pan et al., 2010] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. (2010). Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210.

[Pan and Yang, 2009] Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[Parkhi et al., 2015] Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *British Machine Vision Conference*.

[Parkhi et al., 2012] Parkhi, O. M., Vedaldi, A., Zisserman, A., and Jawahar, C. (2012). Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505. IEEE.

[Rajaraman et al., 2018] Rajaraman, S., Antani, S. K., Poostchi, M., Silamut, K., Hossain, M. A., Maude, R. J., Jaeger, S., and Thoma, G. R. (2018). Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images. *PeerJ*, 6:e4568.

[Russakovsky et al., 2015] 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 (IJCV)*, 115(3):211–252.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Soekhoe et al.,] Soekhoe, D., Van Der Putten, P., and Plaat, A. On the Impact of data set Size in Transfer Learning using Deep Neural Networks. Technical report.

[Song et al., 2019] Song, J., Chen, Y., Wang, X., Shen, C., and Song, M. (2019). Deep model transferability from attribution maps. In *Advances in Neural Information Processing Systems*, pages 6182–6192.

[Tan et al., 2018] Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer.

[Tran et al., 2019] Tran, A. T., Nguyen, C. V., and Hassner, T. (2019). Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1405.

[Vascon et al., 2018] Vascon, S., Parin, Y., Annavini, E., D'Andola, M., Zoccolan, D., and Pelillo, M. (2018). Characterization of visual object representations in rat primary visual cortex. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[Wang and Schneider, 2014] Wang, X. and Schneider, J. (2014). Flexible transfer learning under support and model shift. *Advances in Neural Information Processing Systems*, 27:1898–1906.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

[Zamir et al., 2018] Zamir, A., Sax, A., Shen, W., Guibas, L., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling Task Transfer Learning.

[Zhou et al., 2017] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2017). Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.