UPPSALA
UNIVERSITET

Transfer Learning Techniques on Object Classification

using deep learning and the convolutional neural network

Måns Bengtson, Oskar Holmes, Oscar Holmqvist

Project in Computational Science: Report

February 2022

PROJECT REPORT

Institutionen för informationsteknologi

1 Introduction

Machine learning (ML) has long been a field of interest within both academia and industry, with demand for the flexible and powerful solutions that ML provides is steadily increasing. Large companies and organizations are capable of generating massive amounts of data that can feed this development.

Answering the question "What is machine learning?" is a lot more complicated than one would assume. One could simplify it with an all-encompassing description, as Arthur Samuel does. Samuel claims that machine learning gives computers the ability to learn without being explicitly programmed. One could argue that it is easier and possibly more educational to give examples of machine learning in use, and allow the reader to come to an intuitive understanding on their own. Aurélien Géron illustrates this well by going through development steps of a spam filter.

A classical programmer's approach would be something along the lines of studying spam emails to find similarities between them, such as certain words like "free" or "prize", and write a code to flag these as spam. With more complex problems the programming becomes quite extensive with large amounts of rules. A machine learning algorithm would do the same task, but could, with relatively little effort from the developer, deepen the studies of emails to create a much larger and more complex set of rules. Géron [1] points out that machine learning may reveal relationships that the developer would never have seen.

This may be a good enough explanation when it comes to classical ML, with unfathomable amounts of data-generation and funds with which to pursue training of large and incredibly complex and specialized networks. Within smaller companies and new fields of research however, the access to data may be more limited. This creates a problem, as the success of machine learning to a large extent is determined by the amount of data. Thus, the effort to decrease the need for large sets of data is essential for enabling new applications of machine learning. Transfer learning (TL) offers a simple yet powerful solution to this problem by allowing for transfer of knowledge across domains and tasks. In its simplest terms it utilizes "knowledge" from large well-trained networks which are retrained with much less data to fit new tasks. This technique unlocks the potential to use machine learning in fields with limited data. This project aims to study TL and attempt to find general use cases where TL can be advantageous.

2 Purpose

The purpose of this report is to investigate transfer learning, and when it works well. There is reason to suspect different behaviour when transferring across different domains, but this report is limited to looking at image classification using the domains found in the Cifar-10 data set. The report investigates the performance of the model when transferring from different domains, compared to the baseline performance without TL. The performance is evaluated for a range of data set sizes in combination with different numbers of frozen layers. This result is intended to give some insight in how to effectively apply transfer learning when working with small data set sizes.

3 Theory

This report does not assume the reader to be familiar with machine learning. Thus some of the core concepts are explained here, with focus on the ones with importance to the implementation described in this report. For a deeper understanding we refer to one of many textbooks written on the topic, such as [2].

3.1 Machine Learning

Machine learning is a set of algorithms that makes use of statistical data to improve on a task. Common usage is classification and regression. There are a wide range of algorithms used within the field of machine learning, of varying complexity. Arguably the most successful subset of methods in machine learning is the field of deep learning.

3.2 Deep learning and neural networks

Deep learning is concerned with artificial neural networks, usually referred to simply as neural networks. A neural network consists of a set of partially connected neurons, or nodes. The connection is directed and each connection of two neurons has a weight assigned to it, which determines the degree of activation of the neuron to a given input. This is metaphorically similar to how [3] describes neurons in the human brain– "neurons who fire together wire together". The basic idea of the neural network is to minimize some given loss function by optimizing the weights of the network. In the case of supervised machine learning, minimizing the loss function correlates to improvement on performing a task (regression or classification) on the training data. This is done with the hope that the performance on the training data will be generalized to new data.

The simplest example of a neural network consists only of fully connected layers. In this architecture each neuron in one layer has a weighted connection to every neuron of the following layer. An example of this is shown in Figure 1. A dense layer can be expressed on vector-matrix notation as a function of the previous layer, according to equation (1). Here a x_i is an n dimensional vector, and x_{i+1} and b are m dimensional vectors, and A is an $m \times n$ dimensional transformation matrix from R_n to R_m. The activation function is denoted by σ The layers between the input and output layers are referred to as hidden layers.

$$x_{i+1} = \sigma\left(Ax_i + b\right) \tag{1}$$

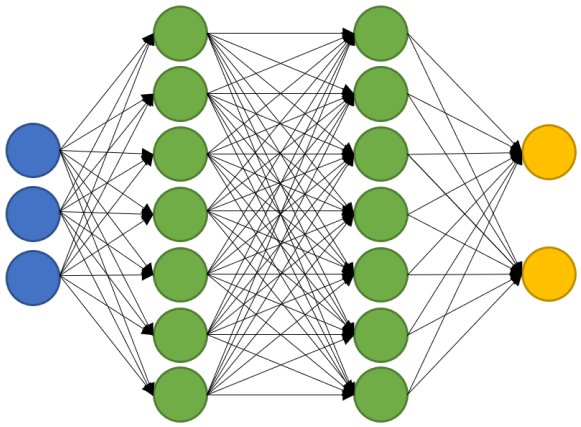

Figure 1: Schematic image of a fully connected neural network

3.2.1 Gradient descent

Gradient descent is a first order optimization algorithm, and is used in machine learning to minimize the value of the loss function as a function of the parameters of the neural network. The idea of the algorithm is to take a step in the direction of the negative gradient, such that the cost function approaches a local minimum. While the principle is simple enough, formalizing this problem mathematically for a larger neural network is not straight forward. Luckily the gradient can be computed analytically with the help of *backpropagation*.

Backpropagation is used to calculate the gradient of the cost function with respect to the neuron activation and the input. It is problematic to compute the gradient with large training data sets, since it can be very computationally expensive. To increase the training speed smaller *batches* of training samples can be used to approximate the gradient. While this only gives an approximation, it is computationally less expensive and can still converge faster. Furthermore an approximated gradient can help the model avoid prematurely converging towards a low performance local minimum. A batch size of 1 is referred to as stochastic gradient descent (SGD). Note that a local minimum for the training data does not necessarily correspond to a local minimum for other data.

3.3 Object classification

A common tasks that is often solved using machine learning is object classification. This task corresponds to classifying the input into one or multiple different possible categories. An important application of object classification the sub-field of image classification, which involves classifying an object represented in a digital image. Such tasks appears in computer vision, surveillance software and text-readers among many other areas. A difficulty arises in the fact that figures inside the picture can be in different parts of the picture as well as of different sizes. The following section presents the most common type of machine learning architecture used to deal with these difficulties.

3.4 Convolutional neural networks (CNN)

Section 3.1 describes one of the simplest architectures of a neural network– the fully connected neural network. While this model is straightforward and works well as a platform to understand more complex architectures, its capability is limited. A large focus of this report is its application in image analysis, and for this purpose the convolutional neural network (CNN) outperforms fully connected models. A CNN is different from a fully connected model in the setting that each neuron only takes input from a certain region in the previous layers. In image analysis, this allows some neurons to be responsible for detecting certain local features of an image, like shapes, colors, or patterns. Mathematically, each layer acts as a convolution between a kernel and the previous layer. The convolution operation is defined in equation (2). The output of convolution is referred to as a feature map. This makes sense considering that the output is the original image with a series of filters applied to it, effectively highlighting different features.

$$\omega * A = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega_{dx,dy} A_{(x+dx),(y+dy)} \tag{2}$$

3.4.1 Hyperparameters

The dimension and function of each hidden layer is determined by several parameters, commonly referred to as hyperparameters. These include the stride, the padding, the kernel size and the number of filters. The kernel size is one of the characteristic hyperparameters of the CNN. It determines how large the filter applied to the previous layer is. In the image classification case the dimensions specify how many pixels are filtered with each movement. Connected to the kernel size is the stride. The stride determines how many pixels are skipped by the kernel when moving, both horizontally and vertically. For example, if the stride is $(2, 2)$, the center of the kernel moves over every other pixel. If the stride is smaller than the dimensions of the filter some pixels are filtered more than once, if it is large some may not be filtered at all.

If the dimensions of a kernel do not line up properly with the image one may have to utilise so called *padding*. A common way of implementing padding is zero-padding. It adds lines of zeroes along the edges of the images to allow the filters to start at an earlier element with the edges included, even if the dimensions would cause the filter to end up "outside" of the image.

3.4.2 Convolutional layer

The convolutional layer is the most important feature of a CNN. The layer's purpose is to enhance low level features into high level features for the following hidden layer. A low level feature can be seen as a filtered image, whereas high level features are encoded low-level features. Géron mentions how this fact is the reason that CNNs are so well suited for image recognition, since it is common for real world images to have this same "hierarchical structure", see [1].

In convolutional layers we do not have a one-to-one connection, because each neuron is connected to all pixels in its "receptive field" on the image or layer. The field is commonly referred to as a kernel, and is a rectangle with several pixels connected to a single neuron in the convolutional layer [1].

3.4.3 Activation function

The neural network makes use of activation functions to map its output. In equation (1) the activation function is denoted by σ. With a linear activation function the output layer can be re-written as a function of the input layer on the form $\mathbf{x}_{out} = \mathbf{A}\mathbf{x}_{in} + \mathbf{b}$. This linear expression limits our ability to build deep networks that accurately capture complex, non-linear relations in the data. Luckily there are multiple activation functions that can be used to avoid this problem. The choice of an activation function is not trivial, as each comes with its own advantages and disadvantages.

The activation function at the last layer should be determined by the purpose of the neural network. For a regression problem the final activation function should allow the output to be in the desired range, which can be achieved with e.g. a linear activation function. For a classification problem, the output value of each neuron should be in the range [0,1], which can be achieved with activation functions such as sigmoid or tanh.

After some epochs of training, it is likely that most neurons have values close to the upper or the lower boundary of the output range. When using a function like sigmoid or tanh, the gradient may become very small. This is referred to as the *vanishing gradient problem*. This causes the network weights to update very slowly, and thus convergence rate decreases. This is referred to as the saturated gradient problem. This problem can be solved by using an activation function like the rectified linear unit (Relu). This function is described in equation (3), and has the advantage of being linear in segments, which gives it a gradient that is constant for values on either side of 0. Hence, using the Relu, can remedy the vanishing gradient problem.

$$ReLu(x) = max(0, x) \tag{3}$$

3.4.4 Pooling layer

Another common element of the CNN is the availability of so-called pooling layers. A pooling layer reduces the number of parameters by reducing the dimensions of the feature map, and therefore decreasing the computational cost of training the model. A pooling layer is created by sliding a kernel over a feature map, and applying a function $\mathbb{R}^{nxn} \rightarrow \mathbb{R}$ to those elements, e.g. taking the average value. The most common pooling layer is the maxpool, where only the maximum neuron output value is taken from a given neighbourhood. This reduces the sensitivity to high frequency components such as Gaussian noise. As the pooling is done over a region, it also decreases the sensitivity to distortions such as rotation and translation. This is because rather than learning the exact position of different features, it looks for the most important feature in a region.

3.4.5 Dropout

During the training process, the dependence on some features might become overemphasized. This means that the model is over-fitted on particular features, a problem that could decrease the models performance. One technique to combat this is to use dropout. This means that during the training process, a certain fraction of the neurons are randomly set to 0. This forces the model to become less reliant on a particular set of features, as after dropout it cannot rely on those feature always being present. Another benefit of the dropout is the decrease in number of parameters when training the model, which helps speed up the training process.

3.4.6 Batch Normalisation

When training the network, the gradient descent algorithm faces the problem of the input for each layer constantly changing as a consequence of the previous layers changing simultaneously. This problem increases as the network becomes deeper. This phenomenon is referred to as the *Internal Covariate Shift* and is defined as the *change in the distribution of network activations due to the change in network parameters during training.* [4] shows that by normalising the distribution of the output feature map the training process will speed up. The technique for doing so is referred to as batch normalisation, and works by normalising the distribution of the input map such that each input feature map has the same mean and variance.

3.5 Transfer Learning

Transfer learning aims to utilise existing knowledge to increase performance in a new task or domain. When attempting transfer learning the intention is to improve learning of the target goal by improving baseline performance, by increasing improvement rate and by improving the final performance. This can better be illustrated in Figure 2. In practice, transfer learning on a neural network is done by initialising the weights of the network with weights from a different network that has been trained on another domain or task, as is explained in [2].

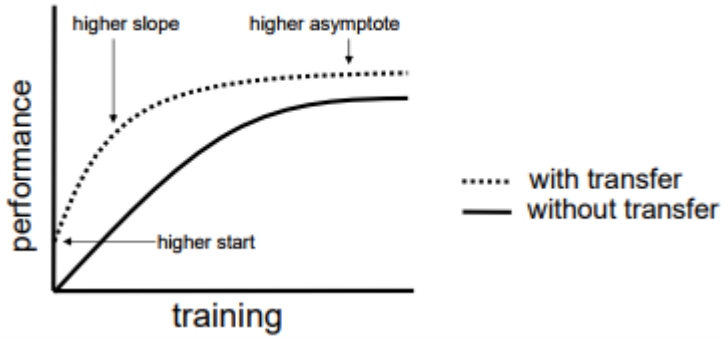

Figure 2: possible benefits of using transfer learning, source: Sarkar, Dipanjan [2]

When intending to implement some kind of transfer learning, one must answer the following three questions:

- What to transfer?

It is not likely that applying the knowledge in its entirety without adapting the solution to the task will give a good result. Thus, one must determine which portion of the knowledge to apply and which part to adapt to the new task.

- When to transfer?

It can be shown that transferring knowledge sometimes gives a worse result than not doing anything, this is known as *negative transfer*. Thus one must determine when not to attempt transfer learning and when it is advantageous.

- How to transfer?

The third and final question, how do we actually transfer the knowledge? There are many different existing algorithms and techniques and they correlate to different results when implemented in different fields and on different tasks [2].

3.5.1 General vs specific features

Although the network is often seen as a black box, the intuition behind transfer learning methods on deep neural networks is that the early layers find general features such as lines, edges and color blobs whilst later layers find specific features such as heads, cars or humans. Thus, the idea is that the early layers are very general and can be applied to any domain and still manage to extract the information necessary for later layers to categorise it. Say the first few layers determines there's a bright circle, then the later layers can be either trained to use that information to determine that it is a car, or to determine it is a sun. In the paper "how transferable are features in deep neural networks?" Jason Yosinski [5] argues that if the first-layer features are general and the last-layer features are specific, there must be a transition between the features somewhere in the network. When it comes to deep transfer learning using neural networks this task of how to determine how many of the earlier layers are related to general features and thus shared between the source domain and the target domain poses a real challenge. A relevant sub-question raised by Yosinski is whether the transition from general to specific occur at a

single layer or over several. Perhaps most of the transition is in a certain layer, but that the previous layer affects the specific features too and thus must be slightly trained.

3.5.2 Fine-tuning

One may recall that when applying transfer learning to neural networks the general idea is to re-train the later layers in the network whilst keeping the earlier layers locked. This is due to the concept that early layers correlate to general features whilst the later layers correlate to more domain-specific features. However, what if the early layers are good at finding the features, but not optimal for the new domain? Perhaps a small training on the early layers would make the network much better at the new given task? This is the intuition behind *Fine-tuning*, which means that one unlocks a part of or the entire network and trains it with a limited data set and a low learning rate [2].

3.5.3 Feature-extraction

One of the most widely used transfer learning methods using deep neural networks is *feature extraction*. Here, the concept is to take a pre-trained model and replace the last layer, the layer tasked with classifying the input, with a new untrained layer that is then trained whilst keeping the rest of the neural network locked. The idea behind this is that the pre-trained neural network has been taught to extract features that are shared between domains or tasks and thus these earlier layers in the network can give these features as inputs into the final layer. This final layer can therefore quickly be taught to classify the image from these features it is given. For example, if the task is to determine whether there is a car or a boat in an input picture, one can use a neural network trained to determine between cats and dogs, replace the final layer that outputs cat or dog with a new layer that outputs car or boat and then train this with a limited amount of data in the target domain [2].

4 Implementation

This report seeks to explore the usefulness of transfer learning, and how it is best utilised to improve performance on classification tasks. We recognise that this may vary greatly from domain to domain, and from task to task, but we still wish to see if any generalisations of TL can be made. The image set used to train our networks is Cifar-10, which has ten image classes or domains.

4.1 Cifar 10

Cifar-10 is one of the most widely used image data sets in the world and consists of ten non-overlapping categories of equally many pictures. The pictures are small (32x32 pixels) with low resolution, making the training less computationally expensive but at the cost of containing less high frequency information. Even humans struggle with categorising some of the pictures. We tried classifying 309 images from the testing set, and this gave an human accuracy of 93.2%.

In order to add additional training samples, augmented instances of the training images are added to the training set. The images are augmented with Gaussian noise and they are 'mirrored', i.e. the rows of the image are reversed. This increases the number of samples that the model is trained on, and is especially important when transferring a model to another domain with few training samples. While only two augmentations techniques are implemented, additional benefits are expected to come with more augmentations.

4.2 'Freezing' layers

Central to our methodology is the concept of 'freezing' layers. Consider a fully-trained network with pre-determined weights and sufficient accuracy in a certain image-classification task. In our project, in order to identify the roles of the varying layers in image classification we vary which layers are 'allowed' to re-train on the new classification task. We 'freeze' up to nince of our ten trainable layers, meaning their predetermined weights are locked and unchangeable, while all 'unfrozen' layers receive training for the new task. The classification layer is always left unfrozen and randomly initialized. This methodology is illustrated in Figure 3.

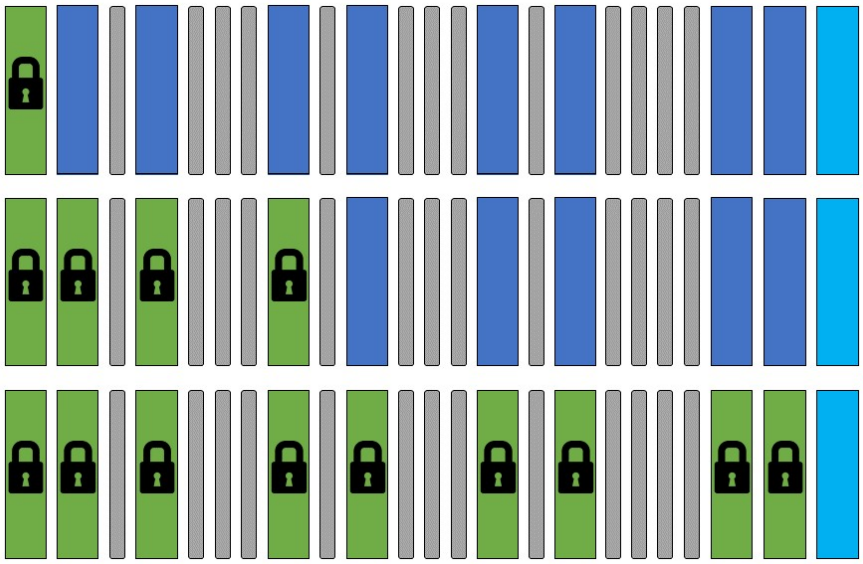

Figure 3: Graphic of 'frozen' layers. Blue layers are trainable, green are frozen, grey are by definition always untrainable and cyan is the classification layer.

4.3 Models

To examine the benefits of transfer learning, three different models are tested. The models are trained on the following sets of classes: cats and dogs, ships and planes, and the set of all classes except cars and trucks. The aim is to see if the choice of domain affects the ability to transfer the model within a given task. This is expected to highlight what attributes that are utilised in transfer learning, as well as in vestige if similarities between domains play a role in the effectiveness of transfer learning. To extend the investigation, we also test the performance when transferring the MobileNetV2 model. This model is trained on Imagenet, a data set of 14 million images with relatively high resolution [6]. MobileNetV2 with pre-trained weights is available in the *Keras.applications* library. The process of implementing TL with the MobileNetV2 model is the same as implementing TL with the other models. The classes cars and trucks from the Cifar-10 data set are used as targets for transfer learning. The performance of the transfer is measured via the achieved accuracy on the target domain of trucks and cars.

4.4 Architecture

The models in each of the source domains are identical to the model in the target domain, with the exception of the output layer. This allows all the weights to be transferred, with the exception of the output layer. All the techniques discussed in the theory section above are utilised in order to improve performance. The architecture is outlined in Table 1. The convolutional base is intended to act like a feature extractor. It is worth mentioning that we have trained and evaluated several models, and this specific architecture is chosen for its performance. This model is in no way a benchmark on the Cifar-10 data set, and the performance could be increased. However, this is not necessary in order to demonstrate the benefits of transfer learning. Only the relative improvement is needed. Additionally, another model with no dense layers (except the classification layer) has also been tested. The architecture of this model is outlined in Table 2. The remastered model is included in an attempt to increase the performance, and investigate if the model could compete with MobileNetV2.

Table 1: Original neural network architecture , with number of weights per layer

Layer type	Dimensions	Trainable parameters
Convolution	32 x 32 x 3	0
Convolution	32 x 32 x 32	896
Batch normalisation	32 x 32 x 32	128
Convolution	32 x 32 x 32	9248
Batch normalisation	32 x 32 x 32	128
Max pooling	16 x 16 x 64	0
Dropout	16 x 16 x 32	0
Convolution	16 x 16 x 64	18 496
Batch normalisation	16 x 16 x 64	256
Convolution	8 x 8 x 64	36 928
Batch normalisation	8 x 8 x 64	256
Max pooling	8 x 8 x 64	0
Dropout	8 x 8 x 64	0
Convolution	8 x 8 x 128	73 856
Batch normalisation	8 x 8 x 128	512
Convolution	8 x 8 x 128	147 584
Batch normalisation	8 x 8 x 128	512
Max pooling	4 x 4 x 128	0
Dropout	4 x 4 x 128	0
Flatten	2048	0
Fully connected	256	524544
Fully connected	128	32896
Output	1	129

Table 2: Remastered neural network architecture, with number of weights per layer

Layer type	Dimensions	Trainable parameters
Convolution	32 x 32 x 3	0
Convolution	32 x 32 x 32	896
Batch normalisation	32 x 32 x 32	0
Convolution	32 x 32 x 32	9248
Batch normalisation	32 x 32 x 32	128
Max pooling	16 x 16 x 64	0
Dropout	16 x 16 x 32	0
Convolution	16 x 16 x 64	18 496
Batch normalisation	16 x 16 x 64	256
Convolution	16 x 16 x 64	36 928
Batch normalisation	16 x 16 x 64	256
Max pooling	8 x 8 x 64	0
Dropout	8 x 8 x 64	0
Convolution	8 x 8 x 128	73 856
Batch normalisation	8 x 8 x 128	512
Convolution	8 x 8 x 128	147 584
Batch normalisation	8 x 8 x 128	512
Max pooling	4 x 4 x 128	0
Dropout	4 x 4 x 128	0
Flatten	2048	0
Output	1	2049

4.5 Hyperparameter sweep

One hyperparameter to be set is the number of frozen layers in the transferred model. Is there a correlation between this parameter and performance on different dataset sizes? This question is central for efficient transfer learning. Therefore, the result section will be mainly focused on the effect of freezing sections of the transferred model. The different values of the freeze hyperparameter is tested on different data set sizes, to show if the optimal number of frozen layers changes with the size of the data set. As previously outlined, one goal of transfer learning is to improve performance on small data sets. Therefore, in order to investigate the performance of the models it is important to provide an overview of accuracy for different data set sizes. Here we use $n \in [10, 20, 50, 100, 200, 400, 800, 1500, 2000]$, where 10 is a very small data set, and 2000 is a relatively large one. These parameters are to be tested on the previously mentioned models. This will hopefully outline any performance differences when transferring from different domains. Transferring between similar domains is expected to give better performance.

5 Results

Here we present the results obtained in this project. Figures 4, 5, and 6 correspond to the parameter sweep for each of the three source domains presented in section 4.3 and consists of four sub-figures. The left side figures show the accuracy of the network whilst the right side figures show the percentage improvement when running the transferred network compared to the reference untrained network. The top sub-figures show the run without using fine-tuning whilst the bottom sub-figures show the run when applying fine-tuning. Observe that the color-scales differ between figures.

Then, the result of running the same experiment using the *remastered* architecture is shown in Figure 7 using the same setup. Lastly the result of using *feature extraction* on the MobileNetV2 network is shown compared to a untrained reference network in Figure 8.

6 Discussion

While the results do not show a consistent optimum across different domains, it still highlights some trends in performance. The use of transfer learning significantly improves the results for many smaller data sets. While the performance changes depending on source domain, overall the best improvement seems to happen in the mid-range data set sizes. The results also seems to suggest negative transfer on very small data sets, but due to the noisiness in performance for smaller samples this is hard to determine. The results also suggests that the performance difference with and without TL decreases as the number of data points become large. This is not unexpected as more data points in the new domain makes the data in the old domain an increasingly smaller fraction of the total training data. With the assumption that the network performance will increase with more data, it is reasonable to conclude that freezing more layers will negatively impact the performance if the available training data is large enough. As a result performance increase diminishes for large n, something that can be seen in most of the simulations. This is suggested by the diminishing performance increase for $n = 2000$. However, a larger range of values for data set sizes would have to be investigated for this conclusion to be solidified.

Comparing the achieved results to the expected results from Figure 2 one can see that the expected higher starting accuracy is only apparent at specific nodes within the heatmaps, mainly when freezing many layers. This is probably due to the network being allowed to focus all the re-training on these last and most impactful layers. The second expected improvement from transfer learning, the higher end asymptote, cannot be seen in the parameter sweeps. The last expected improvement however, the higher slope, is to different extents clearly visible in most domains and number of frozen layers, with the exception of when almost all layers are frozen.

When doing the same comparison on the feature extraction, Figure 8, one can see all of the three expected improvements clearly visible.

The results also indicates that fine-tuning has a mostly positive effect on the results, mainly in the mid-range. This is contrasted with a decrease in performance for small data sets with most of the layers frozen, which seems to indicate that freezing most of the layers works best when the data set is small.

One must note however, that there is a lot of noise when training small samples, and that the result is still unsatisfactory for small data sets.

While there was a clear improvement when transferring from the different Cifar-10 domains, the performance was never better than of mobileNetV2, utilising feature-extraction. This result suggests that this approach should be attempted first, as it is easier to implement, faster to train, and generally gives better or equal performance. A drawback with large networks is the increased memory usage for storing parameters, but with only 14 Mb this should not be a problem for most implementations.

7 Conclusions

The main conclusion to draw is that transfer learning ***may*** improve performance, but can also hamper accuracy with increasingly small datasets. One cannot simply implement transfer learning carelessly and expect an improvement.

One can see the largest increase in performance when using transfer learning with dataset sizes of 100-400 images. This is most definitely specific to our case– architecture and datasets. It is very difficult, perhaps impossible, to be able to say in general how much data is ideal for transfer learning.

Feature extraction is a strong and efficient tool for machine learning when applied to large and well trained networks. It should be attempted first, and only if this does not achieve the requirements one should attempt other transfer learning methods.

Certain domains transfer better to each other than others. This is probably due to the domains being more closely related to each other, but in order to determine this one must first find a way of determining the proximity between domains. Due to the high complex probability density in the high dimensional image feature map, it is inherently difficult to determine this by any conventional means.

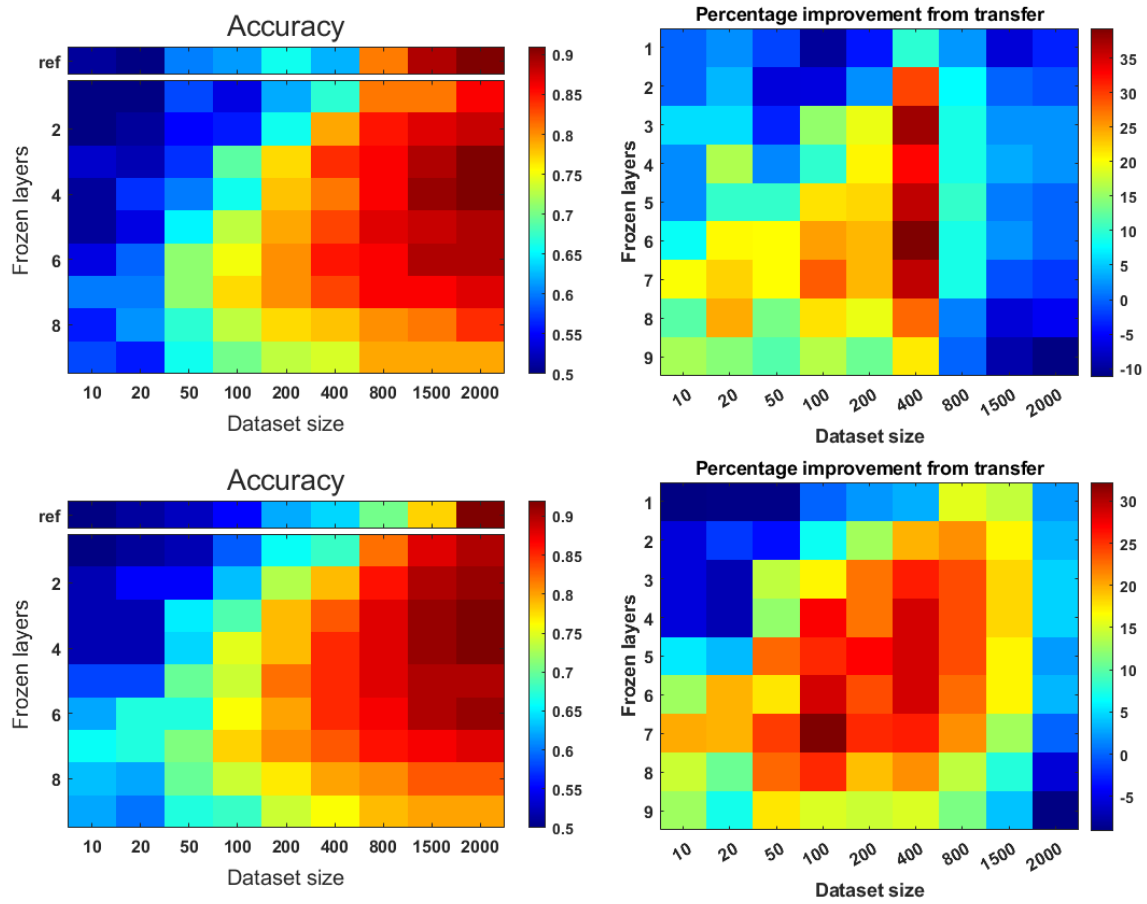

Figure 4: A heatmap of the accuracy and gains from transfer learning with source domain: All. Upper layer is without fine-tuning, lower layer with finetuning, left side showing accuracy and right side showing gains

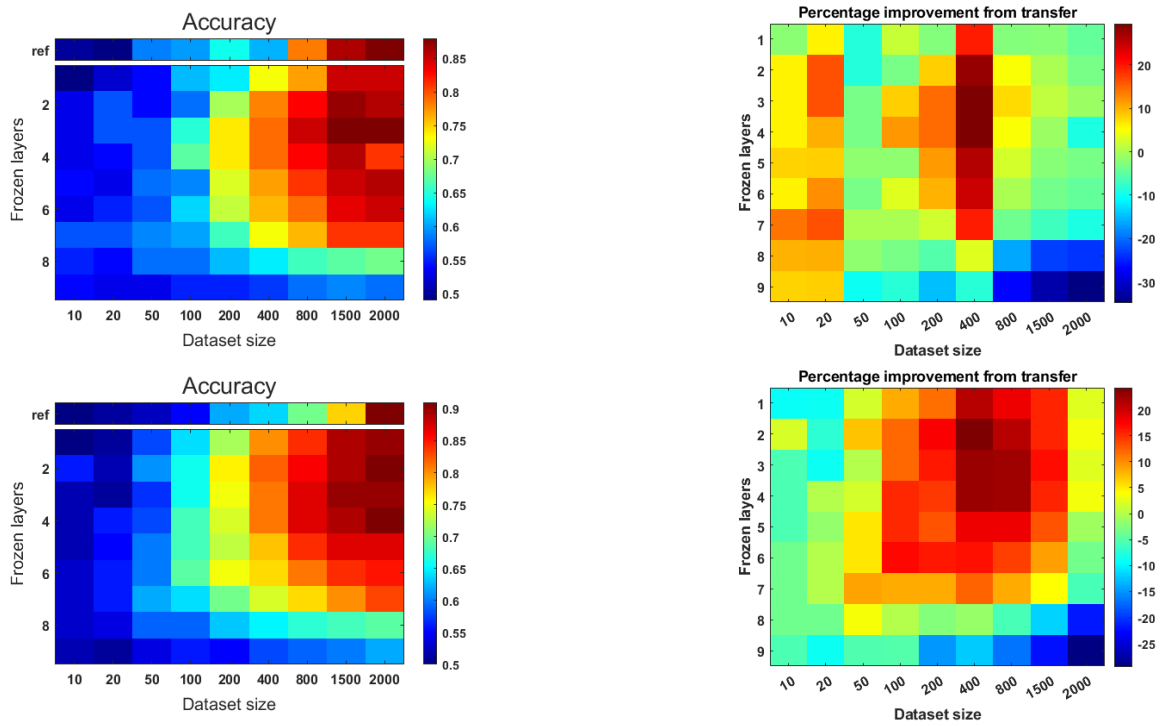

Figure 5: A heatmap of the accuracy and gains from transfer learning with source domain: Dogs and Cats. Upper layer is without fine-tuning, lower layer with finetuning, left side showing accuracy and right side showing gains

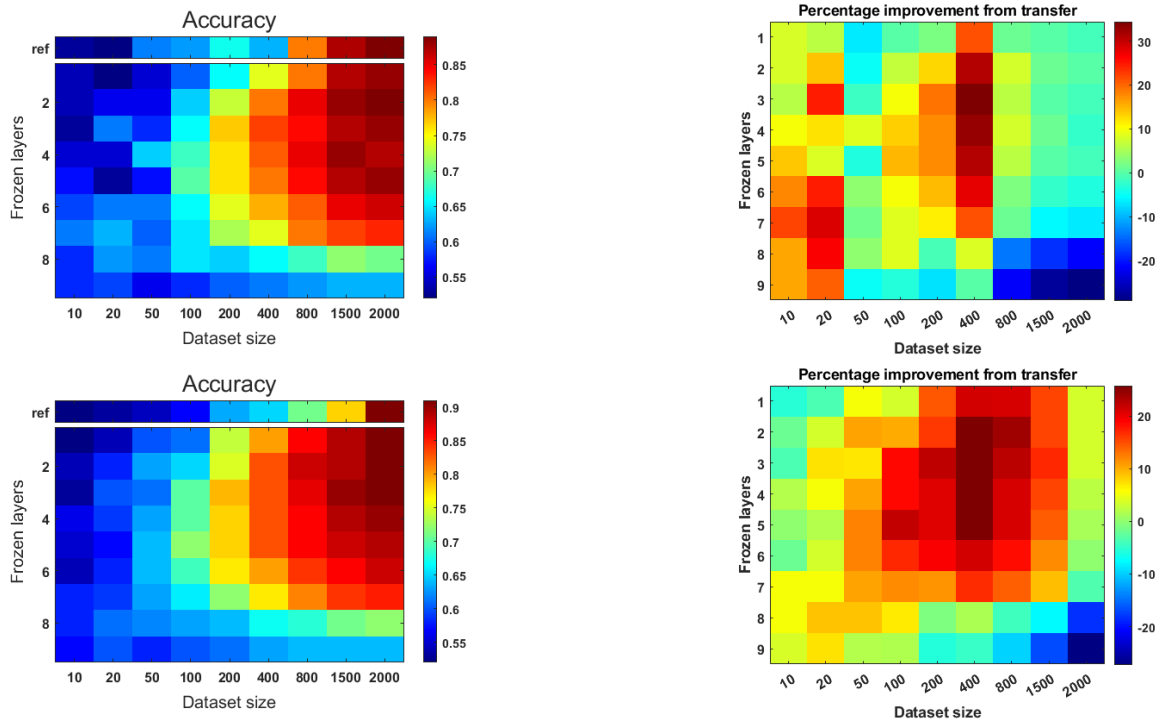

Figure 6: A heatmap of the accuracy and gains from transfer learning with source domain: Ships and Planes. Upper layer is without fine-tuning, lower layer with finetuning, left side showing accuracy and right side showing gains

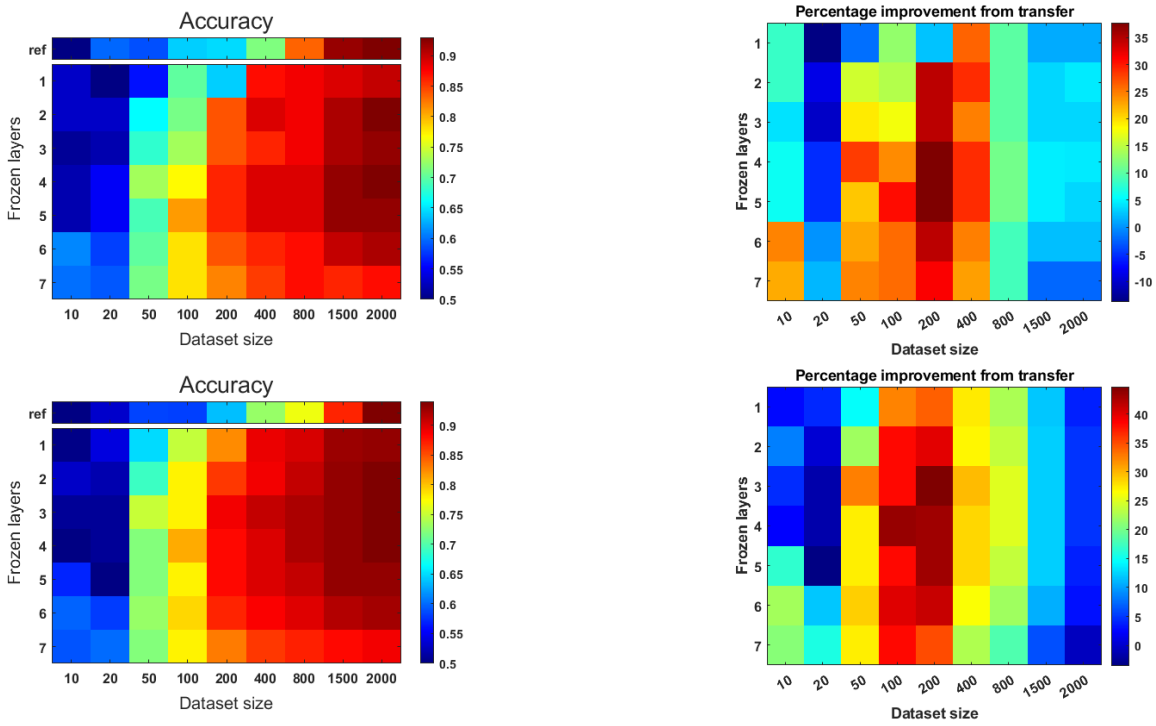

Figure 7: A heatmap of the accuracy and gains from transfer learning with remastered architecture with source domain: All. Upper layer is without fine-tuning, lower layer with finetuning, left side showing accuracy and right side showing gains

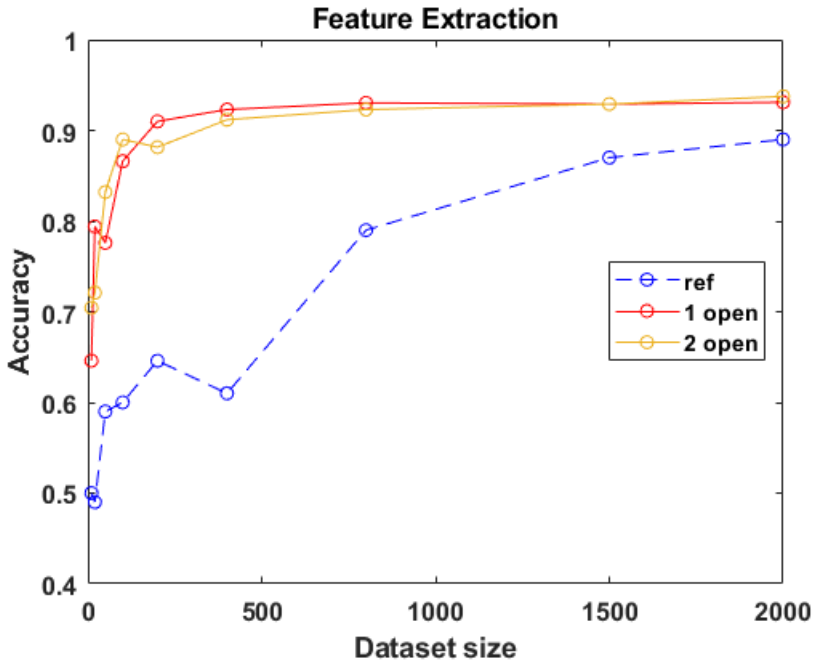

Figure 8: Performance of the well trained architecture MobileNetV2 onto our domain with and without freezing.

References

[1] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow*. by O'Reilly Media, Inc., 2019. ISBN: 9781492032649.

[2] Dipanjan Sarkar. *Hands-On Transfer Learning with Python: Implement advanced deep learning and neural network models using TensorFlow and Keras*. Packt Publishing, 2018. ISBN: 9781788831307.

[3] D.O. Hebb. *The Organization of Behavior*. Wiley & Sons, 1949. ISBN: 9780415654531.

[4] S. Ioffe and C. Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[5] Jason et al. Yosinski. *SHow transferable are features in deep neural networks?* arXiv:1411.1792, 2014.

[6] Jia Deng, Wei Dong, Richard Socher, et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.