NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4173 - Project

A Survey on Data Augmentation and Transfer Learning on Caltech101

Group:
Deep Learning Group 9

Authors:
Magnus Dyre Moe (magnudyr)
Eivind Heldal Stray (eivindhs)
Jahn Patrick Nitschke (jahnpn)

November 26, 2020

Abstract

Convolutional neural networks (CNNs) have revolutionized image classification since the introduction of AlexNet in 2012. Networks such as these are trained on a dataset with virtually no constraints in the amount of data. However, smaller datasets are a common problem for most, as the access to large and reliable datasets is a privilege that only few possess. Datasets with a limited sample size are more challenging when training CNNs, but measures can be taken to improve performance. This paper will investigate two well-known approaches to this problem, namely data augmentation and transfer learning. The overall consensus of these methods suggests a enhanced performance when training a CNN, where this paper also aims to reproduce this result. To do this, the paper is divided into two parts.

In the first part, we implemented a simple 6-layer CNN and experimented with training with and without augmented data. As this is a common technique applied on small datasets, we expected less overfitting and a generally better performance. This was true, although the difference manifested mostly in more consistent metrics for training and validation throughout the training phase.

In the second part, we modified the output layer of a VGG16 net pretrained on the large ImageNet dataset, which contains a subset of quite similar classes, to fit the 102 classes of Caltech101. Training was then done only on the output layer, resulting in a generally better performance than the 6-layer network. Using data augmentation for the training phase of this network yielded negative results, suggesting that data augmentation cannot be done thoughtlessly in order to achieve the benefits associated with this method.

A podcast has been made explaining some machine learning concept in simple terms. `https://drive.google.com/drive/folders/1Z5Sik9inDIYbdI0ogzpJuD9iHeKEqZo7?usp=sharing`

Table of Contents

List of Figures

5figure.caption.6

List of Tables

1 Introduction

In today's increasingly autonomous world, image classification is a natural topic of interest. Ever since 2012, the task of image classification and field of computer vision are subjects that have been dominated by the use of deep learning techniques. Here, we focus specifically on the use of convolutional neural networks (CNNs) for image classification, as CNNs have revolutionised and outperformed traditional computer vision techniques. [Mahony et al., 2019][Shorten and Khoshgoftaar, 2019]

Yet, a broadening need to classify more complex images with great accuracy leads to the development of deeper networks. With deeper networks arises an age-old problem well associated with machine learning: the need for large amounts of data [Shorten and Khoshgoftaar, 2019]. This requirement of big data sets a constraint on the use-case of CNNs, where access to significant amounts of reliable data is a frequent problem pertinent to most people [Perez and Wang, 2017]. As a result of this, a focus on image classification of a limited dataset – and methods to tackle this – is a interesting topic which this paper focuses on.

There are many potential datasets that are of limited size, where an appropriate dataset explored in this paper is Caltech101. In this dataset, there are classes with a particularly few number of images, being as low as 31. This creates problems for CNNs, such as overfitting, which will be discussed further in Section 3.

Consequently, in attempt to tackle the problem of a small sample size, this paper will investigate two techniques to improve the image classification performance on this dataset, namely data augmentation and transfer learning. To provide a quick insight to these methods, data augmentation, in simple terms, refers to the manipulation of data with the main objectives of enhancing the sample size and quality of the dataset [Shorten and Khoshgoftaar, 2019]. Further, as Zhuang et al. [2020] mentions, transfer learning is the aim to leverage knowledge from a related dataset to improve the learning performance or minimize the number of labeled examples required in a target dataset. Hence, in regard to the aforementioned problems, data augmentation provides a means of increasing our sample size, while transfer learning can be understood as the use of a previously trained CNN being applied on Caltech101. Exactly how these methods are implemented is explained further in Section 4.

With the overall motivation, problems and methods for this paper introduced, we can finalize the overall objective of this paper: namely to investigate the use of data augmentation and transfer learning on the classification of images in the Caltech101 dataset. In this context, our paper compares the classification performance of a simple CNN, against CNNs which first incorporates data augmentation and transfer learning, before one that combines both these techniques.

All implementations and code can be found on the github repo of this project: `https://github.com/magnusdyremoe/caltech101.git`.

2 Related Work

Image classification is a task well defined in machine learning, being an area where CNNs have dominated since the introduction of *AlexNet* by Krizhevsky, Sutskever and Hinton in 2012 [Krizhevsky et al., 2017]. Since then, the recent developments in image classification methods can be matched with performances in the *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [Russakovsky et al., 2014]. From this competition, the state-of-the-art methods of *GoogLeNet* and *VGG-16* in 2014, *ResNet* in 2015 and now *FixEfficientNet-L2* in 2020, have been developed, where top-1 accuracy has risen from 69.8% to 88.5% in the classification of ImageNet images. [Szegedy et al. [2014]; Simonyan and Zisserman [2014]; He et al. [2015]; Touvron et al. [2020]]

Moreover, before deep learning, image classification was already an established research area for traditional computer vision techniques, where new research such as in Mahony et al. [2019] involves the combination of deep learning and these traditional techniques. Here, research is motivated by addressing the disadvantages of each side's methods, where for example, the requirement of considerable data in CNNs is a well-known issue. As a result of this, there exists significant amounts of research which delve into methods to mitigate this issue.

To begin, one of the methods to tackle datasets with limited sample size is through data augmentation. This technique is well documented in current research, such as in the papers of Shorten and Khoshgoftaar [2019] and Perez and Wang [2017]. The overall aim of data augmentation is to increase the amount of training data using only information in our training data, as a means of reducing overfitting, as described in Section 4. Developments in data augmentation have also been extensive in recent years, with new techniques being developed every year since 2014 [Shorten and Khoshgoftaar, 2019]. Data augmentation could be done in an iterative process as in Perez and Wang [2017], yet with a limitation on time, our paper does not focus on optimising the data augmentation method, but rather a survey on its general effect on performance.

Transfer learning is another method used to tackle small datasets, and has become increasingly popular. In a survey on transfer learning, Zhuang et al. [2020] summarises, "transfer learning shows some advantages over traditional machine learning such as less data dependency and less label dependency." Further, Zhao [2017] states that the use of transfer learning achieves significant and accurate improvements in for datasets of limited size. Together, these statements are reflective of transfer learning's usefulness as a established deep learning technique, particularly for small datasets.

In regards to research on small datasets such as Caltech101, there exist numerous tasks and competitions that deals with similar problems on data domains such as Kaggle. Otherwise, this dataset has been subject to testing for new research, or a way for researchers to compare their new techniques. For example, *Wide-ResNet-101* by Kabir et al. [2020] has scored 97.11% in terms of top-1 accuracy on the Caltech101 dataset, attaining state-of-the-art performances in this dataset. Here, they have also investigated the use of transfer learning as we intend to in this paper. In addition, as Shorten and Khoshgoftaar [2019] illustrates, there exists plenty of data augmentation methods, of which the Caltech101 dataset has also been used. An example of this is in the paper by Taylor and Nitschke [2017], where generic data augmentation techniques are used to improve CNN performance on several datasets, including Caltech101.

With the popularity of CNNs rising, the constraint on the use-cases of this method is constantly being challenged. This is observed from the abundance of research in this field, resulting in various established techniques such as data augmentation and transfer learning to improve CNN performances on small datasets. Hence, our paper focuses to survey these deep learning techniques, with the goal of reproducing the aforementioned increases in performance from these methods. However, this paper is somewhat unique in the fact that we assess baseline performance increases on a standard un-tuned CNN, whereas most research focus on either surveying or developing state-of-the-art techniques.

3 Data

3.1 The Caltech101 dataset

The Caltech101 dataset consists of 9146 RGB digital images of objects, split into 101 different object categories and Google background images. The size of the images is in close vicinity of 300x400 pixels with small variations, and was developed by the California institute of technology in 2003 to facilitate computer vision and image classification [L. Fei-Fei and Perona, 2003]. These images are not uniformly distributed but range from 31 to 800 images per category, which can be visualized in Figure 1. The low sample size and class imbalance makes for some difficulties when training a CNN on the dataset, where its limited size and imbalance is associated with overfitting and performance [Buda et al., 2017]. Here, overfitting describes a model that performs well on training data but badly on validation data. As mentioned previously, this is a problem that arises from small datasets, and can be solved through methods such as adding more training data through data augmentation [Rao et al., 2018]. In this paper, the focus will be on addressing the size of the dataset rather than the imbalance.

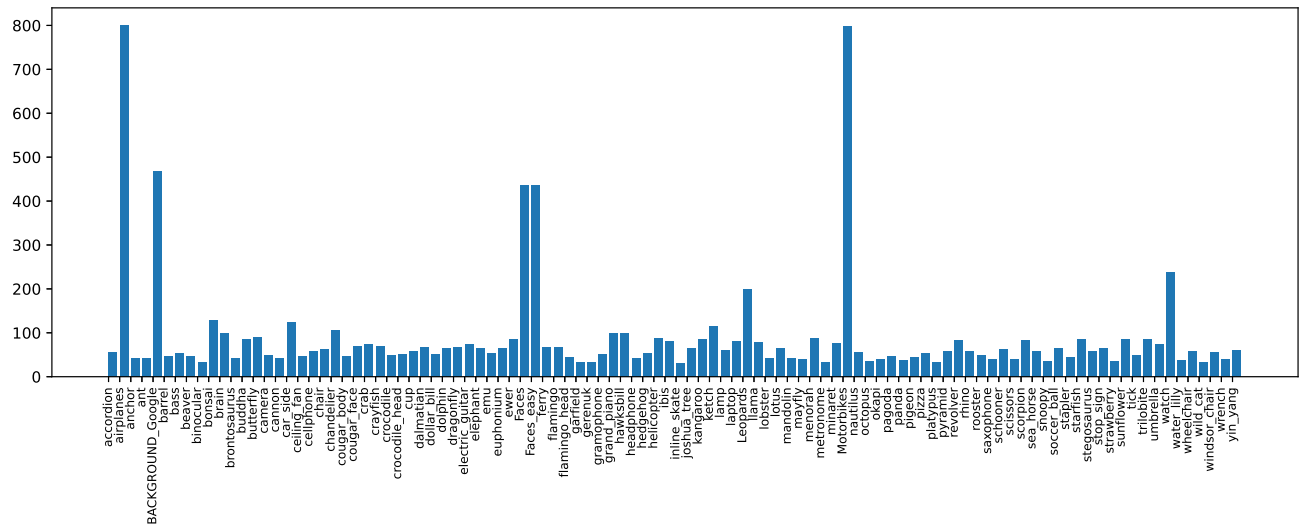
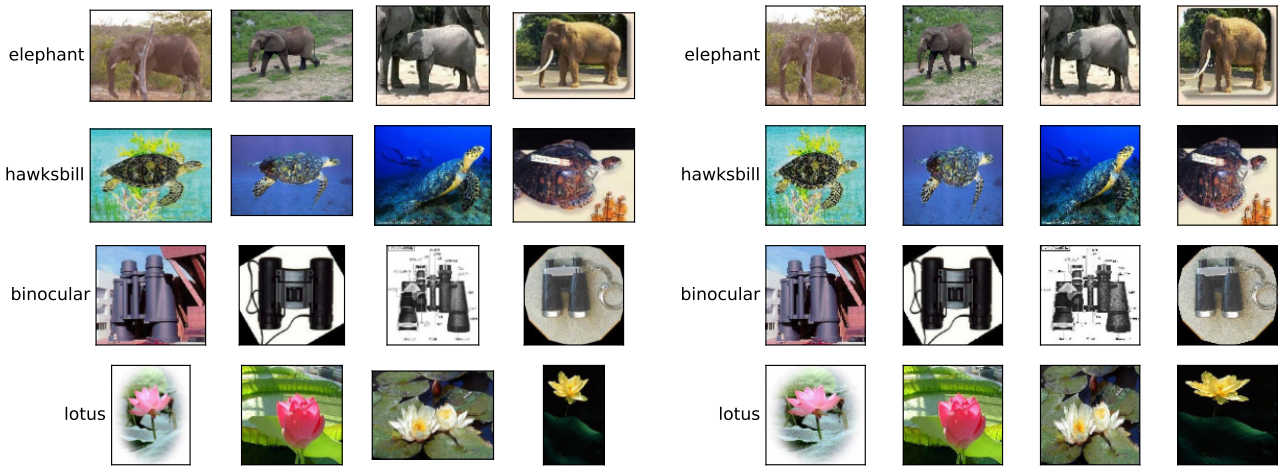


Figure 1: Image distribution and classes of the Caltech101 dataset, with very few images in some classes and a notable class imbalance. Plot created by *data_visuals.py*, found on Github.

By browsing through the categories seen in Figure 1, we observe that the majority of images are quite uniform, with image objects presented similarly in terms of orientation and size. Furthermore, there is seemingly little clutter[1] which allows for easier detection of features. Some images from the dataset are illustrated in Figure 2.



(a) (b)

Figure 2: 4 images from 4 classes in the Caltech101 dataset. Figure 2a shows the original images, while Figure 2b shows the images resized. As observed, the images are relatively uniform in size, presentation and orientation. To elaborate, the images are roughly the same size, the objects are located near the center of the image and the objects have the same orientation (elephants facing left). Plot created by *data_visuals.py*, found on Github.

As the images are uniform in size and presentation, they are not necessarily representative for practical inputs, which one might expect for later use. In more practical inputs, images may be more prone to clutter, with greater variance in display, presentation and orientation. The uniformity in the data could possibly allow for object detection based on the mean of a class, rather than distinct features, which is unrealistic. In practice this means that an airplane may be classified correctly based on the blue sky in the background and the length and width of the aircraft relative to the image, rather than distinct features like the wings, tail and tube-like shape.

As mentioned above, when the sample size is relatively low, such as in Caltech101, the model is prone to overfitting. To explain this, overfitting occurs when the model fits the training data quite well, but there is a substantial discrepancy in accuracy between training and validation. The model then does not generalize and while it may perform well on the training images this may not be applicable on the validation and testing set. For these kind of datasets data augmentation can be a useful tool to reduce this effect if adding more data is not an option.[Rao et al., 2018] This will be further elaborated in Section 4.

3.2 Train/validation/test split

The Caltech101 dataset can be found on multiple open-source websites, such as Kaggle. The data is pre-structured when downloaded, with the 101 object classes and background images separated into different folders containing the respective images. Before implementing a CNN, the data must be split into training, validation and test data. This was done using the *os* library default in *Python*. Here, we defined a probability for images being used for training, validation or testing. Each image was then defined with a 75% probability of being used for training, 15% probability of being used for validation and 10% probability of being used for testing. After running the script on the data the result was 6782 images for training, 1464 images for validation and 898 images for testing. This equates for 74.17% of the images for training, 16.01% of the images for validation and 9.82% of the images for testing. The result of this split were organised sub-folders of images in training, validation and testing. The script used for the training, validation, testing, split can be viewed on the github repository for this project found in Section 1.

When implementing a machine learning algorithm each of these datasets are related to a different task. Fittingly, the train data is used to train the algorithm. Through the training process, the model will become a increasingly better fit for the training data. The validation data is also used in the training process. On the contrary to the training data, the validation data does not inflict any changes to the designed model. It is rather used to validate

[1] Clutter in this context refers to noise in the images

whether the model is a good fit for the designated validation data. Lastly, the test data is set aside until after the algorithm is finished with the training process. With this data, performance metrics for the algorithm can be calculated on previously unseen data. The unseen data is important when the model is evaluated after training, as the performance may differ from the validation set.

3.3 Data preprocessing

Through all simulations images are re-scaled, mapping color channels to values between 0 and 1, as it allows for better performance because the images becomes more uniform, neglecting illumination variances. [Bechar et al., 2019].

Also, since images are not necessarily perfectly uniform in size, all images are re-sized to (150, 150, 3) when running the self-made network in section 5.2 as a measure to be able to classify images of different size. Furthermore, for the pre-trained VGG16 network in section 5.4, all images are re-sized to (224, 224, 3) to fit the input layer of the pre-defined network.

Data augmentation is a form of data pre-processing that can be utilized when training a CNN. Because data augmentation is a technique we aim to investigate in this paper, it will be described in section 4.3.

4 Methods

The algorithm that will be used for image classification is a CNN. On Kaggle, 723 datasets are found when searching for "Image classification", and most of the proposed solutions to these seem to be CNN-implementations. Generally, CNN has proven to be the most suitable method for image classification to date as stated in Section 1.

4.1 CNN architecture

CNNs are comprised of 3 general layers – convolution layers, pooling layers and fully connected layers. A layer represents a state in the network. When designing a CNN, these layers form the structure.[O´Shea and Nash, 2015]

4.1.1 Convolution layers

Convolution layers are the characteristic layers of a CNN, giving the network its name. The layer is based on learnable kernels. A kernel is a tensor, often small in spatial dimension, aiming to capture distinguishable features. The kernel moves across the input, calculating the scalar product for each value in the kernel. This trait allows the network to effectively capture specific traits given the spatial position of the input. The result of this is what is commonly known as an activation. [O´Shea and Nash, 2015]

A convolution layer is constructed to choices of hyperparameters known as kernel size, kernel type, stride, padding and activation function. Kernel size is the spatial dimension of the kernel. The kernel type are the scalar values in the kernel which will differ depending on different features extracted. Stride is the movement of the kernel with respect to the input. Padding is a process expanding the input to preserve input dimensionality. Activation function is used to control the output of a layer. It is often a non-linear function aiming to capture non-linearities in the input. [Mohd Aszemi and Panneer Selvam, 2019]

4.1.2 Pooling layers

Pooling layers aims to achieve spatial invariance through reducing the dimensionality of the activation [Scherer et al., 2010]. Alternatively, these layers can be understood as layers that create "downstream" representations that are more robust to variations in data [Lee et al., 2015]. Moreover, by reducting the dimensionality, the amount of parameters and the computational complexity of the model is consequently reduced [O´Shea and Nash, 2015]. The pooling layer operates over all the activations while preserving the depth of the output.

4.1.3 Fully connected layers

Fully connected layers in the network are layers with connections between all inputs and outputs. These layers are often few in comparison to convolution layers, but account for the majority of parameters in a CNN. Alternatively, it is possible to use sparsely connected layers, reducing the optimization problem while maintaining high accuracy [Ardakani et al., 2017]. The role of these layers is to map the extracted features to an output with a dimension equal to the number of classes [Simonyan and Zisserman, 2014].

4.2 Depth of the Network

The depth of a network is also a tuneable hyperparameter. Generally, a shallow network is referred to as a network with few hidden layers. Similarly, a deep neural network then contains many hidden layers. When referring to the depth of a network, the depth corresponds to the number of tuneable layers in the network. This equates to the convolution and fully connected layers as the pooling layers are not tuneable. Increasing the depth of the network will increase both performance of the network and the computational complexity as documented through VGG16 and other deep networks used for image classification.

4.3 Data augmentation

Figure 3: The same image visualized 5 times while being randomly augmented. Data augmentation was implemented through *ImageDataGenerator* from *Keras*, allowing images to shift horizontally and vertically, rotating, horizontal flipping, zoom and shear. [2]

Data augmentation is a term that describes a variety of manipulations on the data. In the case of images, these include manipulations such as rotations, color manipulations, random erasing of segments of the images, brightness manipulation and more. By manipulation of the training data, the dataset is effectively expanded [Mahony et al., 2019]. This has proven to be a method suitable for reducing overfitting and thus improving the performance when training data is limited. [Shorten and Khoshgoftaar [2019]; Rao et al. [2018]]

4.4 Transfer learning

As mentioned in the introduction, Zhuang et al. [2020] describes transfer learning as the aim to leverage knowledge from a related dataset to improve the learning performance or minimize the number of labeled examples required in a target dataset. In our context, transfer learning offers the chance for CNNs to learn with limited data samples by transferring knowledge from models pretrained on large datasets [Wang et al., 2020]. In practice, "transferring knowledge" means to utilise a network with pretrained parameters, where one alters part of the network to fit a desired need.

An underlying assumption is that the model is trained on similar enough data to extract features that are relatable to the dataset in question. Taking for example the VGG16 net trained on ImageNet, one can train the output to fit a dataset like Caltech101. This is due to many of the classes in Caltech101 being represented in ImageNet [Zhang et al., 2018]. The VGG16 network is particularly well suited for transfer learning, whereas networks such as ResNet and GoogLeNet are less popular for transfer learning given the absence of fully connected layers. Using this model, which is trained on a quite similar but larger dataset, namely ImageNet, may yield satisfactory results despite not being entirely trained on Caltech101. [Zhang et al. [2018]; Shorten and Khoshgoftaar [2019]]

[2] Plot created by *augmented_data_visuals.py*, found on the Github repository for this project.

The structure of VGG16 will not be discussed further in this paper, yet the reader is encouraged to look at "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Simonyan and Zisserman [2014].

4.5 Design choices

First, to discuss the network, we decided to use fewer fully connected layers than convolution layers as the majority of parameters comes from the fully connected layers. Furthermore, max pooling was used as it converges faster than other subsampling methods while being superior in selection of invariant features and improve generalization [Scherer et al., 2010].

In terms of optimisation, we decided to optimise the different networks (found in section 5.2 and 5.4) through equal schemes. This was mainly decided through the fact that we wanted to assess the effect data augmentation and transfer learning on CNN performance, without the expectation that the networks should be perfectly tuned. With that in mind, it was decided that both networks should use the same loss function, the categorical cross-entropy function, and the same softmax activation output. The cross-entropy loss has proven to accelerate the backpropagation, the training process, while providing good overall network performance [Nasr et al., 2002]. Furthermore, the loss function was minimized through mini-batch gradient descent, with a batch size of 32, and RMSprop as the chosen optimization algorithm. The default learning rate for RMSprop was used, which is 0.001. RMSprop was chosen as the optimization algorithm even though the Adam optimizer is the more common choice. This was largely due to the the inflicted time constraint from adding momentum and bias correction. [Ruder, 2016]

The only hyperparameter subject to further tuning was the number of epochs, chosen with the aim of convergence in accuracy and loss.

5 Implementations and results

Training a CNN is a time-consuming process. Hence, it is beneficial to implement such a model in a cloud computation environment. As such, the CNN models were trained on a virtual machine on multiple cores which sped up the training process significantly. Using Google's cloud service, along with the use of built in machine learning frameworks such as *tensorflow* and *keras*, the complex algorithms were relatively straight-forward to implement.

With the goal of investigating the effects of data augmentation and transfer learning on a CNN, we set up four individual Python scripts which implemented these methods on a CNN. In these four files included a CNN, though with varying implementations: standard, with data augmentation, with transfer learning, with both data augmentation and transfer learning. Having four independent files kept the the code implementation structured and manoeuvrable. Also, the Python scripts were written as Jupyter notebooks and can be found on this paper's Github page.

When testing a network, different performance metrics are used to evaluate the previously unseen testing data, mentioned in section 3.2. To evaluate the performance of the network, the chosen metrics were accuracy, precision and recall. Commonly used formulations of these are found in equations (1),(2) and (3). Briefly explained, accuracy is all correct classifications divided by all classifications. For the Caltech101 dataset, we expect the accuracy for highly represented classes to generally be good with or without data augmentation. Hence, an increase in accuracy may indicate that the performance on less represented classes gets better. Furthermore, precision is the proportion of correct predictions relative to the *total number of predictions* for a given class. On the other hand, recall is the proportion of correct predictions relative to the total *number of instances* of the class. [Goutte and Gaussier, 2005]. While accuracy is an overall performance metric for the dataset, precision and recall delve more into class-specific evaluation. Research suggests that precision and recall may not be the best suited metrics for performance given that these measures are biased, and may not tell the full story [Powers, 2011]. However, these metrics give a clear insight into class-specific performance and will hence be interesting to use.

$$Accuracy = \frac{True\ Positives}{True\ Negatives} \tag{1}$$

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives} \tag{2}$$

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives} \tag{3}$$

5.1 Implementing data augmentation

In addition to the re-scaling described in Section 3, data augmentation was implemented for the standard and transfer learning CNN models. This was achieved through the ImageDataGenerator function from the Keras Preprocessing library, where, as discussed previously, only the training set would be augmented. Specifically, every image was randomly subjected to a set of manipulations: rotation of up to 40 degrees, displacement in height and width of up to 20%, horizontal flip, zoom up to 10% and shear up to 20%. The full implementation of data augmentations can be viewed in two of the notebooks found on Github.

When using augmented data inputs, images will alternate from epoch to epoch. Hence, it is expected that the networks will require more iterations to fit the data correctly. This was tuned through epochs, whereas the network in section 5.2 uses 50 and 100 epochs for training without and with data augmentation respectively. Furthermore, because training the network used for transfer learning in section 4.4 was increasingly cumbersome, the number of epochs were set to 30.

5.2 A 6-layer CNN

The network was implemented with the Model class in Keras, using a sequential model. The model consisted of convolution layers with 32, 64, 128 and 256 kernels. The kernel size remained the same for every convolution layer: 3×3, with a stride of 1. Between every convolution, max pooling was utilized with kernel size of 2×2. Furthermore, every convolution layer was implemented with the ReLU activation function to capture non-linearities. Finally, the tail-end of the network consisted of a flattening layer with two fully connected layers, with 128 and 102 nodes. The activation functions for these layers were ReLU and softmax respectively, where the softmax activation function requires the number of nodes in the last layer to match the number of classes. This yields a 6-layer network, consisting of 4 tuneable convolution layers and 2 tuneable fully connected layers. To reduce the training time, the input shape was defined as (150, 150, 3) compared to the input shape of (224, 224, 3) used for transfer learning in section 5.4.

5.2.1 Standard data inputs

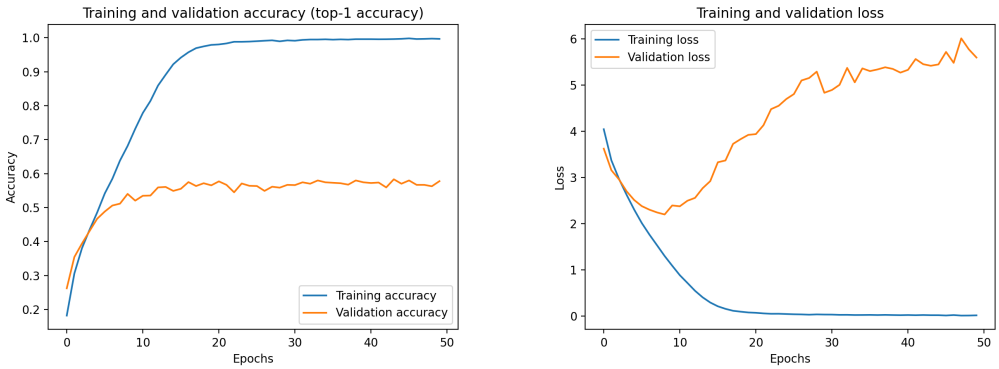


Figure 4: Training history of the 6-layer network without data augmentation inputs

Running the standard CNN on just the Caltech101 dataset resulted in the plots in Figure 4. We observe that the validation accuracy converges after 10-15 epochs while the training accuracy keeps rising. Furthermore, the loss around this time starts increasing for the validation data while the training loss function converges to zero. This behaviour is characteristic of a model that is overfitting the training data, where training data accuracy is improving without improvements to the general performance.

5.2.2 Augmented data inputs

Using the same CNN model with an augmented dataset results in the the plots in Figure 5. Clearly, the discrepancy between training and validation accuracy that was was observed in Figure 4 is no longer as prevalent and we no longer observe an inflection point in the loss. This suggests that the model generalizes better with data augmentation, yet still overfitting after roughly 40 epochs. Also, the validation accuracy did not increase, staying roughly the same as in Figure 4. This is further elaborated on in the next section.

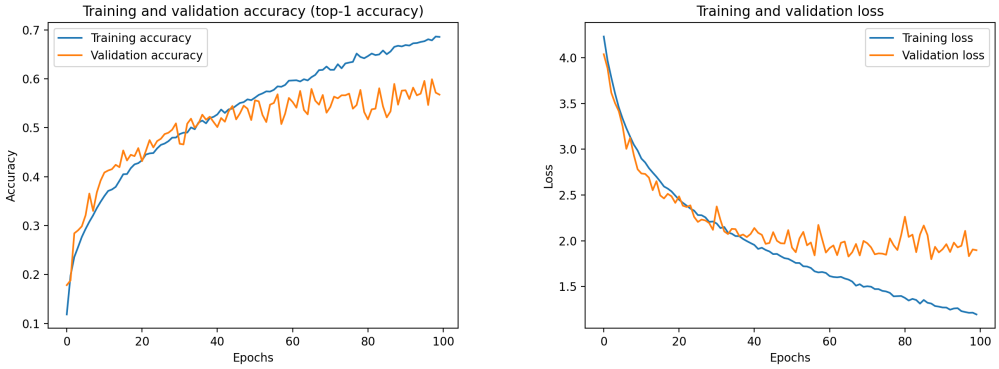


Figure 5: Training history of the 6-layer network with augmented data inputs

Moreover, we also observe more oscillations in the validation accuracy. This is not unexpected as while the training data is augmented for every epoch, alternating how the model fits the training data – the validation data remains unaltered. This type of validation response is common when the learning rate is too high, however, as we use an RMSprop, an adaptive optimizer, this is not the case. Furthermore, this type of response is also less likely with deeper networks as weight changes in individual nodes are less significant.

5.3 Comparison for the 6-layer model with and without data augmentation

Metric	Without augmentation	With Augmentation
Precision	0.60	0.63
Recall	0.56	0.57
Accuracy	0.56	0.57

Table 1: Metrics for trial over test data, found as the weighted average of all classes. Training done with and without data augmentation.

By examining Table 1, we see a tendency where all the metrics score better for the augmented data implementation, with increases of 1% - 3% in these metrics. This was lower than our original expectations given the extensive consensus on the effect of this technique. To try and understand this result, we can consider different factors that could affect performance.

An important factor when evaluating these scores is that there is an imbalance in the test dataset proportional to the class imbalance in Caltech101. This means that by predicting only highly represented classes can potentially yield a quite high overall accuracy despite being sub-optimal for class specific precision. By looking at class specific metrics in table 2, this seems not to be the case as the precision for these classes is high, suggesting that the model is well fit for these classes. In contrast, the precision and recall were lower for the less frequent classes (dalmatian, mandolin and revolver), while still supporting the overall trend of slight improvement when adding data augmentation.

	Metric	Airplanes	Motorcycles	Faces	Dalmatian	Mandolin	Revolver
Without data augmentation	Precision	0.92	0.98	0.84	0.29	0.33	0.54
	Recall	1.00	0.99	0.93	0.50	1.00	0.64
Data augmentation	Precision	0.98	0.95	1.00	0.75	1.00	0.83
	Recall	0.92	0.90	0.98	0.75	0.33	0.45
	Occurrences	87	80	45	4	3	11

Table 2: Precision and recall for the 3 most frequent classes and 3 selected less frequent classes using the 6-layer CNN. Metrics with and without data augmentation

Although these results are promising, other classes vary in whether they are better or worse in these performance metrics. Therefore, this experiment does not conclude better performance with data augmentation apart from consistency in training and validation accuracy and loss. The classification reports can be further inspected in the notebooks on Github.

Furthermore, the networks seem to have reached a limit of performance with 56% and 57% accuracy, independent of the use of data augmentation. Hence, adding data augmentation did not yield a substantially improved performance, as the possibly could not "learn" more despite having more information available from data augmentation. This may correlate to the depth of the network, and while the performance is acceptable compared to random guessing, one could expect a deeper network to yield a better performance, due to more thorough feature extraction and feature mapping. Also, state-of-the-art convolutional neural networks tend to be deeper.

5.4 Transfer learning based on VGG16

In Keras, pre-trained networks are readily available in the applications module of the deep learning library. For transfer learning, we chose to use the VGG16 network, as it is a good fit for transfer learning in our paper. (See Section 4) By manipulating the output layer to fit the Caltech101 dataset the VGG16 network complies with the desired output dimension of 102 classes. Furthermore, defining the output layer as trainable, making all other layers un-trainable is significantly less time consuming than training the entire network. This allowed the network to be trained in 3 or 5 hours, depending on whether data augmentation was used, compared to the 2-3 weeks required to train the entire VGG16 network on ImageNet [Simonyan and Zisserman, 2014].

The implementation can be found on notebooks corresponding to transfer learning (VGG16) on Github.

5.4.1 Standard data inputs

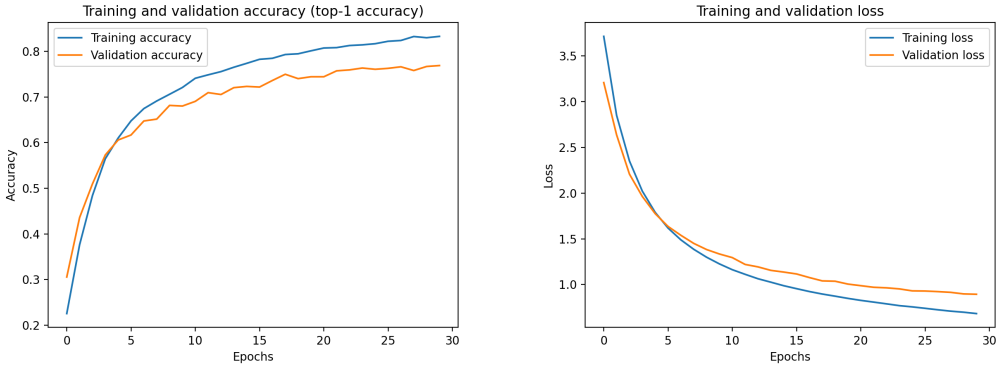


Figure 6: Training history of the modified pre-trained VGG16 network without data augmentation inputs

Judging from figure 6 the model seems to be a perform better on the Caltech101 dataset. The accuracy discrepancy observed in figure 4 is less prevalent, indicating that the model generalizes better for the data. Nevertheless, the model seems to be overfitting after roughly 5 epochs. However, it is worth noting that the difference between training and validation is substantially lower than previously observed in the 6-layer CNN implementation in Section 5.2.

5.4.2 Augmented data inputs

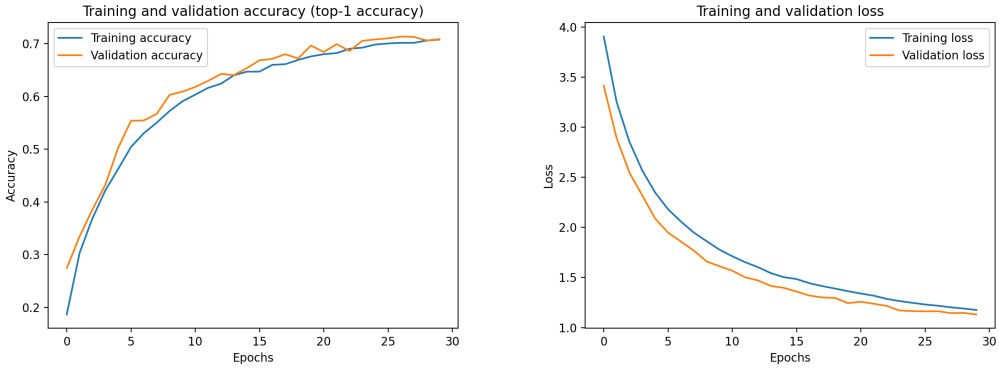


Figure 7: Training history of the modified pre-trained VGG16 network with data augmentation inputs

On the contrary to previously observed training histories, the network in figure 7 seems to be underfitting through the entire training process by studying the loss. However, the loss values for training and validation coincide more towards the end of the training process, indicating that the model is approaching the optimal fit. Any training beyond this point would then result in overfitting. Interestingly, the accuracy has decreased by adding data augmentation. This will be further discussed in the next section.

5.5 Comparison for VGG16 with and without data augmentation

With the modified VGG16 network the performance greatly increased from the 6-layer model as displayed in Table 3. However, there was a decrease in performance when using data augmented inputs for the training of the output layer compared to the approach without data augmentation. This drop in performance seemingly stems from the use of data augmentation, and therefore it seems that reducing overfitting by data augmentation introduces other issues. When training the network, we allowed for re-scaling, rotation, shifting, flipping, zooming and shearing. On the contrary, the pre-trained VGG16 network utilized re-scaling, horizontal flipping and RGB color shifting [Simonyan and Zisserman, 2014]. The difference in augmentations could be the reason there is a drop in performance when using data augmentation with the modified VGG16 network. If we were to utilize more suitable data augmentation for the network we could perhaps expect an increase in accuracy, as the type of data augmentations performed does have implications on CNN performance [Perez and Wang, 2017].

Metric	Without Augmentation	With Augmentation
Precision	0.81	0.76
Recall	0.78	0.72
Accuracy	0.78	0.72

Table 3: Metrics for trial over test data, weighted average. Training done with and without data augmentation.

	Metric	Airplanes	Motorcycles	Faces	Dalmatian	Mandolin	Revolver
Without data augmentation	Precision	0.90	0.98	0.98	1.00	1.00	1.00
	Recall	0.98	0.99	0.96	1.00	0.33	0.82
Data augmentation	Precision	0.86	0.89	0.96	0.80	1.00	1.00
	Recall	0.95	0.90	0.96	1.00	0.67	0.73
	Occurences	87	80	45	4	3	11

Table 4: Precision and recall for the 3 most frequent classes and 3 selected less frequent classes using transfer learning. Metrics with and without data augmentation

From the more frequent classes (Airplanes, Motorcycles and Faces) in Table 4, there is a negative trend in performance for the model trained with data augmentation compared to the one where data augmentation is not used. For the less frequent classes however (Dalmatian, Mandolin and Revolver), the trend is less evident. Since the frequent classes have a negative trend, the overall performance can be expected to be lower as these represent a significant portion of the dataset. Table 3 verifies this trend. The full classification report can be found in the corresponding notebook on Github.

6 Conclusion

Datasets such as Caltech101 are not ideal when training a convolutional neural network because of unbalanced classes, and more importantly limited data. Yet as we have explored in this paper, there are actions that can be taken to enhance performance. These measures include data augmentation and transfer learning using a pre-trained network manipulating only the output layer.

Applying data augmentation on the training data artificially increases the size of the training set by making alterations to the images. This method is commonly used as a preprocessing technique when datasets are limited. The simple 6-layer model presented in Section 5.2 without data augmentation resulted in an accuracy of 56%. Running this network without data augmentation was clearly problematic as the model was fitted to the training data quickly, while not generalizing to other data. This was seen from the divergence between training accuracy and validation accuracy as well as the divergence between training loss and validation loss. Adding data augmentation in Section 5.2 to the 6-layer model did not increase the general accuracy enough to conclude with any better performance, as it only improved by 1% to 57%. However, the gap between accuracy and loss in the training and

validation data was substantially lower. The improved consistency signifies that the model is not overfitting to the same extent as the model without data augmentation and is therefore a positive result.

The transfer learning from VGG16 to Caltech101 assumes data of similar features that can be transferred from ImageNet to Caltech101 by retraining the output layer with appropriate dimensions. The modified pre-trained VGG16 network performed better than the 6-layer model with accuracy for the models trained with and without data augmentation, with overall accuracies of 72% and 78% respectively. For this network, the data augmentation resulted in lower accuracy, which may be the result of data augmentation being used differently on the origin VGG16 network.

The results of this paper are reflective of methods with potential, where it was observed that data augmentations and transfer learning could possibly be powerful tools to tackle the problem of small datasets when utilized correctly. To build on the result of lower performance of the VGG16 network with data augmentations, a natural extension of this paper would be to explore the different possible data augmentations and assess the effect of these when combined with transfer learning.

Another aspect that would have been interesting to address is the imbalance in the Caltech101 dataset, for example either by oversampling the underrepresented classes or by undersampling the overrepresented classes. Furthermore, an interesting experiment could also be to balance classes by adding images from other datasets that contain the same categories, assuming that such datasets exist for all the classes.

The concept of transfer learning with high performance pre-trained CNNs used on small datasets is not unique for this paper. Ultimately, the results are promising, and exploring this further and on other small datasets could yield interesting results. We have demonstrated that it is to some extent applicable when the data is relatable. This may have applications in areas where labeled data is limited and difficult to obtain.

Bibliography

Arash Ardakani, Carlo Condo, and Warren J. Gross. Sparsely-connected neural networks: Towards efficient vlsi implementation of deep neural networks. 2017.

Mohammed El Amine Bechar, Nesma Settouti, Mostafa El Habib Daho, Moudloud Adel, and Mohammed Amine Chikh. Influence of normalization and color features on super-pixel classification: application to cytological image segmentation. 2019.

Mateusz Buda, Atsuto Maki, and Maciej Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106, 10 2017. doi: 10.1016/j.neunet.2018.07.011.

Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. *Lecture Notes in Computer Science*, 3408:345–359, 04 2005. doi: 10.1007/978-3-540-31865-1_25.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

H M Dipu Kabir, Moloud Abdar, Seyed Mohammad Jafar Jalali, Abbas Khosravi, Amir F Atiya, Saeid Nahavandi, and Dipti Srinivasan. Spinalnet: Deep neural network with gradual input, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 2017. doi: https://doi.org/10.1145/3065386. URL `https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`.

R. Fergus L. Fei-Fei and P. Perona. Caltech101, 2003. URL `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`. Last updated: 2006.

Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. 2015. URL `http://proceedings.mlr.press/v51/lee16a.pdf`.

Niall O' Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Adolfo Velasco-Hernández, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. *CoRR*, abs/1910.13796, 2019. URL `http://arxiv.org/abs/1910.13796`.

Nurshazlyn Mohd Aszemi and Dhanapal Durai Dominic Panneer Selvam. Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, 10:269 – 278, 06 2019. doi: 10.14569/IJACSA.2019.0100638.

G.E. Nasr, E.A. Badr, and C. Joun. Cross entropy error function in neural networks: Forecasting gasoline demand. 2002. URL `https://www.researchgate.net/publication/221439058_Cross_Entropy_Error_Function_in_Neural_Networks_Forecasting_Gasoline_Demand`.

Keiron O´Shea and Ryan Nash. An introduction to convolutional neural networks. 11 2015. URL `https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks`.

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

David Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness correlation. 2011.

M.R. Rao, V. Prasad, P. Teja, Md Zindavali, and O. Reddy. A survey on prevention of overfitting in convolution neural networks using machine learning techniques. *International Journal of Engineering and Technology(UAE)*, 7:177–180, 05 2018. doi: 10.14419/ijet.v7i2.32.15399.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. URL `http://arxiv.org/abs/1409.0575`.

Dominik Scherer, Andreas Müller, and Sven Behnke. *Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition*. Springer Berlin Heidelberg, 2010.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL `http://arxiv.org/abs/1409.4842`.

Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation, 2017.

Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet, 2020.

Kafeng Wang, Xitong Gao, Yiren Zhao, Xingjian Li, Dejing Dou, and Cheng-Zhong Xu. Pay attention to features, transfer learn faster cnns. 2020.

Chen-Lin Zhang, Jian-Hao Luo, Xiu-Shen Wei, and Jianxin Wu. In defense of fully connected layers in visual representation transfer. In Bing Zeng, Qingming Huang, Abdulmotaleb El Saddik, Hongliang Li, Shuqiang Jiang, and Xiaopeng Fan, editors, *Advances in Multimedia Information Processing – PCM 2017*, pages 807–817, Cham, 2018. Springer International Publishing.

Wei Zhao. Research on the deep learning of the small sample data based on transfer learning. *AIP Conference Proceedings*, 1864(1):020018, 2017. doi: 10.1063/1.4992835. URL `https://aip.scitation.org/doi/abs/10.1063/1.4992835`.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.

Appendix

A split_train_test.py