Università degli Studi di Milano

Data Science for Economics

Machine Learning Project

Image Classification with Convolutional Neural Networks

by
Jan Philip Richter
Matriculation Number: 20547A

and
Giordano Vitale
Matriculation Number: 14310A

submitted to
Prof. Dr. Nicolò Cesa-Bianchi

January 29, 2024

Abstract

Convolutional neural networks are great machine learning models to deal with the task of image classification. This report introduces multiple model specifications in order to compare their performance on a classification task with the goal of identifying whether an image contains a muffin or a chihuahua.

We propose several different architectural approaches and evaluate their benefits and drawbacks. We first consider two standard sequential architectures, varying in depth and complexity. Moreover, hyperparameter tuning is performed to further optimise one of the models' training phases. The second family of models adopts the VGG framework with the aim to compare whether this already established architecture yields improvements over our previous models. Lastly, a pair of residual neural networks is built to investigate the effect of a vastly deeper and more complex model architecture. All models are compared in predictive accuracy on the training- and validation data. One of the proposed models is selected to compute a more accurate risk estimate via cross-validation.

Additionally, we perform transfer learning to analyse how well the classification performance can be improved when considering pre-trained, state-of-the-art neural networks.

Finally, we discuss our obtained results and propose possible improvements.

Contents

1 Introduction

1.1 Task

The goal of this project is to use convolutional neural networks to build a binary classifier, which can detect if an image contains a muffin or a chihuahua, using the TensorFlow framework.

For this, we experiment with 3 different network architectures and propose a total of 6 models to investigate which approaches yield the most desirable results. The choice of network architecture, its advantages and disadvantages, as well as the fine-tuning of hyperparameters is scrutinised thoroughly in the subsequent chapters.

Finally, cross-validation is used to calculate a reliable risk estimate for one of the models with respect to the zero-one loss.

1.2 Dataset

The dataset[1] considered for this report consists of JPEG images of muffins and chihuahuas, scraped from Google images. There are a total of 5,917 images, divided into a training and a testing set, as shown in the table below:

	Muffin	**Chihuahua**	**Total**
Train	2,174	2,559	4,733
Test	544	640	1,184
Total	2,718	3,199	**5,917**

The entire dataset has been checked for possible corrupted files, as well as duplicates, but none were found. The test set, also referred to as the validation set, will be used to get an estimation of the models' performances on unseen data which allows for better model comparison.

1.3 Data Preprocessing

Preprocessing data is a crucial step, as it enhances important features that improve a neural network's performance. Moreover, it introduces diversity to the training data,

[1]Dataset obtained from: https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification

helping the model generalise better to unseen examples and therefore reducing overfitting. Krizhevsky et al. (2012) perform several preprocessing methods which inspired us to implement the subsequently discussed techniques.

Image Conversion and Resizing

To enable a neural network to take images as input they have to be converted from the JPEG format into a numeric format.

The TensorFlow framework takes the input data in the form of a tensor, which is a multidimensional array of numbers. Our images are thus converted into 3-dimensional tensors, with the 3rd dimension representing the colour channels, red, green, and blue. The values of each pixel of each colour channel range between 0 and 255 and indicate the intensity of the given colour in the corresponding pixel.

The input shape of the tensors has to be identical for any given model. Since our dataset contains images of varying dimensions, we perform a downsizing procedure to guarantee a constant tensor input shape of (224, 224, 3) via bilinear interpolation. This means we end up with all images having a size of 224×224 pixels with 3 colour channels. To make our models more comparable we use the same input shape for every model. All models receive the input images in batches of size 32 during the training run.

Horizontal Flipping

We randomly flip our images around the horizontal axis with a probability of 50%. This is done to ensure that the models are more robust to the orientation of objects, improving their ability to generalise.

Colour Augmentation

A colour augmentation procedure randomly adjusts the contrast of the images on a colour channel basis. For every colour channel the mean value is computed. A random factor determines the intensity of the adjustment on each pixel for that channel using the following formula:

$$x_{c,new} = (x_{c,old} - \mu_c) \times factor_c + \mu_c$$

where μ_c denotes the mean pixel value of colour channel c, $x_{c,old}$ denotes the original value of the pixel in that colour channel and $factor_c$ is a random draw from the uniform distribution $[0.7, 1.3]$[2].

This contrast adjustment introduces altered lighting conditions, which may help highlight and detect patterns in the images better.

[2] https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomContrast

Image Standardisation

Standardisation is a common procedure used to guarantee a more robust and accurate model performance.

Each image is standardised to have a mean value of 0 and a variance of 1. The standardisation is performed on a per-image basis using the following formula for a given image:

$$image_{std} = \frac{image - \mu}{sd_{adj}}$$

where μ denotes the mean value of all elements in the image and $sd_{adj} = \max(sd, 1/\sqrt{N})$ where sd is the standard deviation of all elements in the image and N is the number of all elements[3].

1.4 Implementation

The project is implemented in Python using the TensorFlow framework (version 2.14). The entire code can be found in this GitHub repository. For faster execution time in the training phase of the networks, we utilise the T4 GPU provided by the Google Colaboratory environment.

[3] https://www.tensorflow.org/api_docs/python/tf/image/per_image_standardization

2 Sequential

The first family of models we propose are straightforward sequential-based models.

2.1 Sequential 1

2.1.1 Architecture

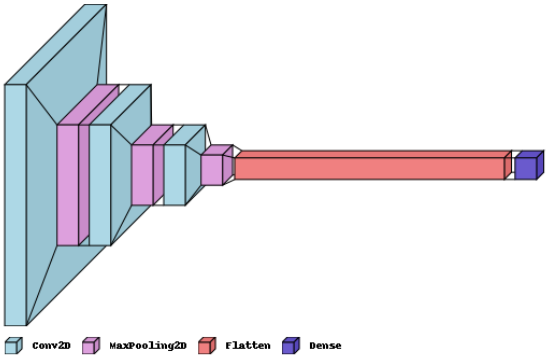
The architecture of the first sequential model is displayed in Figure 2.1 and follows a very simplistic approach. The model consists of 3 convolutional blocks, where every block is made up of 1 convolutional layer with kernel size (3×3), aiming at capturing the images' patterns, followed by a MaxPooling layer for size reduction. The downsampling limits both the computational costs by reducing the number of parameters of the model, as well as the risk of overfitting by helping to extract the main features of the images. The number of channels stays constant at 32 throughout each convolutional block.

Figure 2.1: The convolutional layers are denoted by Conv⟨filter size⟩-⟨number of channels⟩. ReLU activation functions and preprocessing layers not displayed for brevity.

(a) Sequential 1 Architecture

Layer Name	Output size
Input (RGB)	(224, 224)
Pre-processing	(224, 224)
Conv3-32	(224, 224)
MaxPool2, /2	(112, 112)
Conv3-32	(112, 112)
MaxPool2, /2	(56, 56)
Conv3-32	(56, 56)
MaxPool2, /2	(28, 28)
Flatten	25,088
Output (Sigmoid)	1

(b) Sequential 1 Visualisation



The weights are initialised using the Xavier-Normal initialiser, introduced by Glorot and Bengio (2010). Every convolutional layer is followed by a ReLU activation function, which introduces non-linearity in our model. This is a crucial property to prevent the vanishing gradient issue. The output is a fully connected layer, containing 1 neuron with the sigmoid activation function, which provides a mapping in the [0, 1] interval. The obtained result is the network's predicted probability that the respective image belongs

to the class "Muffin"[1]. The Sequential 1 model has a total of 44,481 trainable parameters, making it the by far sparsest model in our analysis. It can thus be considered as the benchmark for further model variations.

2.1.2 Execution Specifications

The loss function we consider for the model's training run is binary cross-entropy. This is a suitable loss function as it not only considers if a label has been correctly classified, but it quantifies the difference between the true label and its predicted probability. Higher prediction uncertainty thus results in a higher penalisation.

The algorithm used to optimise the loss function is Adam which was introduced by Kingma and Ba (2014). Adam takes advantage of two extensions of the classic gradient descent algorithm, namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). This allows the learning rate to be computed dynamically based on past gradients, in contrast to regular SGD, where the learning rate is fixed. The starting learning rate is set to 0.001. All other parameters are left at their TensorFlow-specific default values. The neural network is trained for 50 epochs on the training set and validated on the test set.

Unless stated differently, this execution setup will be used for all further models.

2.1.3 Results

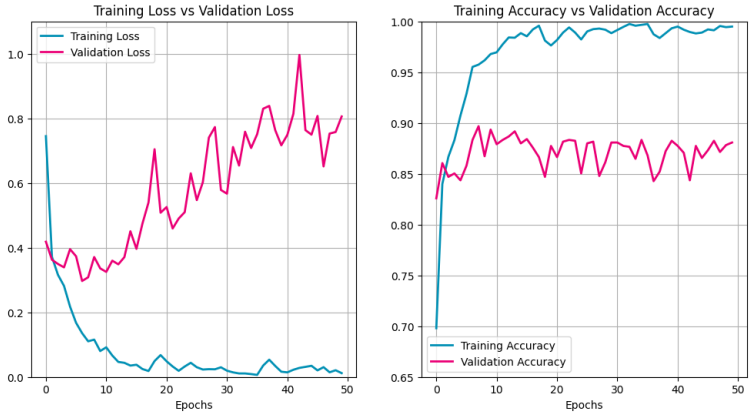
Figure 2.2 shows the performance of the Sequential 1 model on both the training and validation set.

We can observe that the model severely suffers from overfitting. While the training loss converges towards zero over time, the validation loss only follows that trajectory for a short time. After 5-10 epochs the validation loss starts to diverge from the training loss and follows an upward trend. This means, that the patterns, that the model learns on the training set, do not generalise well on new unseen data points. The uncertainty in predictions produced by the model rises over time and at the end of 50 epochs a validation loss of circa 0.8 is reached.

The increased validation loss does not affect the accuracy of the classification too much. The validation accuracy starts to plateau after 10 epochs and stays at a level between 85%-87%.

[1] A predicted probability of above 50% thus results in the image being classified as a muffin, while a probability of below 50% means the network classifies the image as a chihuahua.

Figure 2.2: Sequential 1 Results



2.2 Sequential 2

2.2.1 Architecture

The Sequential 2 model adds several augmentations to the benchmark Sequential 1 model, having an increased depth and complexity, as well as various additional specifications. This deeper and more complex setup aims at improving both the predictive accuracy of Sequential 1, as well as reducing the tendency of overfitting.

The model retains similar structural specifications to its aforementioned counterpart, though it is made up of 5 convolutional blocks, instead of 3, as can be seen in Figure 2.3. Each convolutional block now consists of a convolutional layer, an AveragePooling layer and a BatchNormalisation layer.

The number of channels per convolutional layer has been increased and does not stay constant anymore. We move from convolutional layers with 128 channels in the first 2 blocks, to 64 channels in the subsequent 2 blocks and finally 32 channels in the last block. The idea is to encourage the detection of as many patterns as possible in the early stages of the model and reduce this increased complexity over time.

MaxPooling has been substituted by AveragePooling, which serves a similar purpose as previously. The dimensions of the tensor are halved over time with the main goal of preventing overfitting.

One of Sequential 2's major differences from its shallower counterpart is the addition of BatchNormalisation, introduced by Ioffe and Szegedy (2015). BatchNormalisation is a regularisation method that has proven highly useful in many state-of-the-art neural network architectures and is thus often preferred over alternatives such as dropout layers. BatchNormalisation normalises the features by estimation of the 1st and 2nd moments within each batch. This scaling allows for a faster convergence of the gradient descent, thus making the training phase quicker. However, the more interesting property of BatchNormalisation is the reduction of the internal covariate shift, which makes the

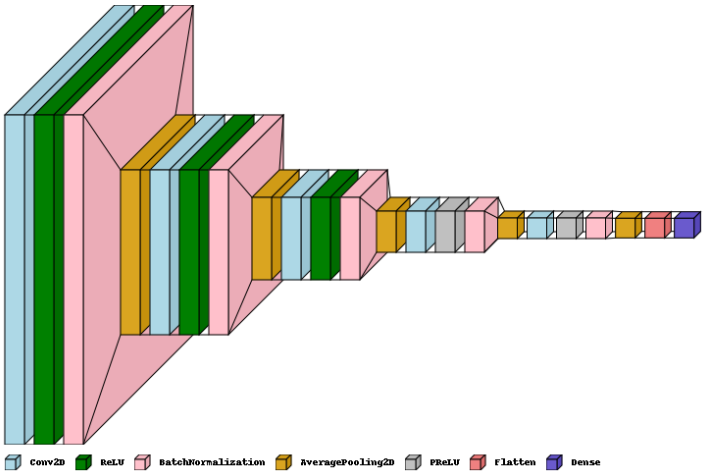
training process more stable, resulting in a reduced tendency to overfit[2].

Figure 2.3

(a) Sequential 2 Architecture

Layer Name	Output size
Input (RGB)	(224, 224)
Pre-processing	(224, 224)
Conv3-128	(224, 224)
BatchNorm	(112, 112)
AveragePool2, /2	(112, 112)
Conv3-128	(112, 112)
BatchNorm	(56, 56)
AveragePool2, /2	(56, 56)
Conv3-64	(56, 56)
BatchNorm	(28, 28)
AveragePool2, /2	(28, 28)
Conv3-64 (PReLU)	(28, 28)
BatchNorm	(14, 14)
AveragePool2, /2	(14, 14)
Conv3-32 (PReLU)	(14, 14)
BatchNorm	(7, 7)
AveragePool2, /2	(7, 7)
Flatten	1,568
Output (Sigmoid)	1

(b) Sequential 2 Visualisation



Two further alterations are the utilisation of the parametric-ReLU (PReLU) activation function and the He-Normal initialiser, both introduced by He et al. (2015). The PReLU activation function introduces the tunable parameter α_i in the following form:

$$f(y_i) = \begin{cases} y_i, & \text{if } y > 0 \\ \alpha_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

The additional parameter allows for tuning of the slope in the negative part of the function. The replacement of ReLU with its parametric counterpart has shown to result in improved classification accuracies, by introducing relatively little more complexity. The number of additional parameters introduced by PReLU is equal to the number of channels, which is negligible in comparison to the overall number of weights in the model[3].

[2]It is noteworthy that the reduction of the internal covariate shift by BatchNormalisation is debatable. For instance, Santurkar et al. (2018) propose that BatchNormalisation does not improve the distributional stability, albeit they uncover other benefits of BatchNormalisation, like a smoother optimisation landscape during the training phase.

[3]However, the overall number of weights in this model has significantly increased, totalling 340,033. This is down to the vastly increased depth.

The He-Initialiser draws numbers from the following distribution to initialise weights:

$$N\left(0, \sqrt{\frac{2}{n^l}}\right)$$

where n^l denotes the number of neurons n in layer l. This initialisation method has been proven paramount when initialising weights in a setup, where the ReLU or PReLU activation functions are in use. Where the He-Initialiser allows for deep neural network architectures to converge, the previously used Xavier-Initialiser does not.

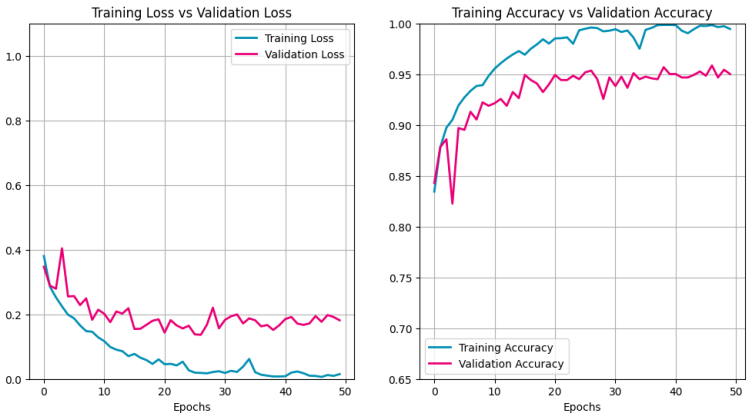
2.2.2 Hyperparameter Tuning

For the Sequential 2 model, we introduce hyperparameter tuning for the learning rate of the Adam optimiser, intending to additionally optimise the training run of the network. We experiment with 6 distinct possible learning rate values in the interval $[0.01, 0.0005]$. The tuning phase consists of a total of 10 different executions with 10 epochs each[4], using Bayesian Optimisation to find the optimal value. The learning rate obtained through this procedure (learning rate = 0.0005), is used in the final training run of the model.

2.2.3 Results

The results of Sequential 2's training run, displayed in Figure 2.4, show a significant improvement in both loss and accuracy curves.

Figure 2.4: Sequential 2 Results



We can observe that the overfitting behaviour of Sequential 1 has almost completely been solved with the augmentations made for this model. Most notably the validation

[4]Meaning some of the possible learning rate values are tested more than once.

loss does not diverge from the training loss anymore. The overall loss value has also been reduced by a factor of 4 (from 0.8 for Sequential 1 to 0.2 for Sequential 2). The predictive accuracy has also been considerably improved, reaching slightly above 95% on unseen data.

We believe that this major performance improvement is mostly caused by the regularisation effect of the BatchNormalisation layers and the more suitable initialisation procedure of the He-Initialiser.

3 VGG

The next class of models we propose follows a VGG-style[1] architecture. This model architecture for convolutional neural networks was originally introduced by Simonyan and Zisserman (2014) and yielded significant performance improvements in comparison with previously used architectures in image recognition tasks. In fact, this was the new state-of-the-art architecture back then and placed 1st and 2nd in the ImageNet 2014 competition, which was the most prestigious image classification competition in the world at that time. Their main contribution was significantly increasing the network's depth up to 19 convolutional layers while keeping the filter size very small, using (3×3) convolutional filters.

We introduce a pair of models which follow this architectural setup, albeit in a shallower form for computational simplicity, to investigate if this approach yields performance improvements compared to the straightforward sequential models, introduced in the previous chapter.

3.1 VGG 1

3.1.1 Architecture

The idea behind the architectural setup of the first VGG model is a sequence of convolutional blocks, whose last layer is a MaxPooling layer with stride 2, to half the dimensions of the tensor along the way. This is not a fundamentally different setup compared to the aforementioned straightforward sequential models. The main difference is that the number of filters in the convolutional layers is doubled each time after the MaxPooling has halved the tensor size. Another change is the fact, that the VGG-setup considers multiple convolutional layers to follow one another subsequently, which can be found in the third convolutional block of the VGG 1 model. Lastly, there is a fully connected layer with 8 neurons included before the output neuron.

Simonyan and Zisserman (2014) state that the subsequent convolutional layers yield two main advantages in comparison to other approaches. A sequence of convolutional layers with a relatively small filter size of (3×3) is computationally less expensive, as the number of tunable parameters is still smaller than the number of weights coming from a single convolution with a larger filter size of (7×7) for instance[2].

[1] VGG denotes Visual Geometry Group, which is the group name from the Department of Engineering Science at Oxford University, whose scientists developed this architecture.

[2] A sequence of 3 convolutional layers with a (3×3) filter size and 16 in- and output channels has $3 \times (3^2 \times 16^2) = 6,912$ weights while a single (7×7) convolution has $7^2 \times 16^2 = 12,544$ weights.

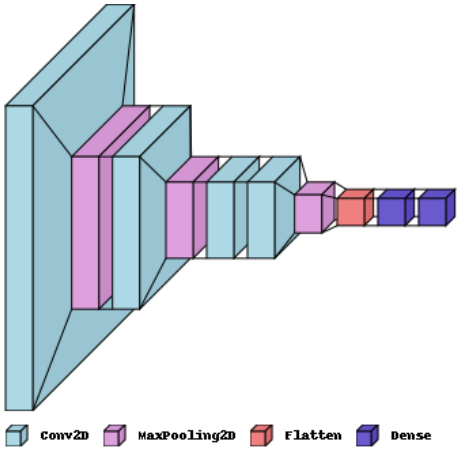
Additionally, every convolutional layer contains another ReLU activation function adding further non-linearity, which has the benefit of increasing the "discriminant" power of the decision function. The total number of tunable parameters is 461,937.

Figure 3.1

(a) VGG 1 Architecture

Layer Name	Output size
Input (RGB)	(224, 224)
Pre-processing	(224, 224)
Conv3-16	(224, 224)
MaxPool2, /2	(112, 112)
Conv3-32	(112, 112)
MaxPool2, /2	(56, 56)
Conv3-64	(56, 56)
Conv3-64	(56, 56)
MaxPool2, /2	(28, 28)
Flatten	50176
Dense	8
Output (Sigmoid)	1

(b) VGG 1 Visualisation



3.1.2 Results

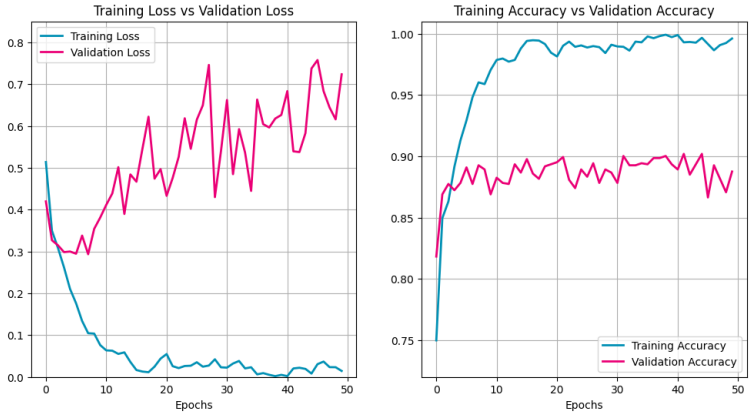
The results of the training run of the VGG 1 model are shown in Figure 3.2. We can observe that the model quickly starts overfitting, as the training and validation loss take on different trajectories after the 10th epoch. A similar pattern was also visible in Figure 2.2, which displays the results of Sequential 1. While the training loss converges towards 0, the validation loss shows an upward trend and high volatility, which are both undesirable features.

The overfitting is also visible in the deviation between training and validation accuracy, with the training accuracy converging towards 100%. However, this behaviour does not affect the model's predictive ability too much, as the validation accuracy plateaus after roughly 10 epochs at 87% and doesn't fluctuate much after that.

The plateau of the validation accuracy in combination with an increasing validation loss means, that the model still classifies the same percentage of pictures correctly over time, but with higher uncertainty.

One possible explanation for the decline in accuracy compared to the Sequential 2 model in the previous chapter could be the model's shallowness and lack of regularisation layers, like BatchNormalisation or Dropout, which were not originally considered by the inventors of this architecture.

Figure 3.2: VGG1 results



3.2 VGG 2

3.2.1 Architecture

The VGG 2 model introduces our deepest model so far with a total number of 12 weight layers resulting in a total number of 715,761 trainable parameters. Besides the increased depth, the most significant changes are the increased number of subsequent convolutional layers and the addition of (1×1) convolutions in the last 2 convolutional blocks.

(1×1) convolutions, which were previously used by Lin et al. (2013) in their "Network in Network" architecture, are essentially a linear projection onto the space of the same dimensionality. However, as previously mentioned Simonyan and Zisserman (2014) describe that those convolutional layers introduce further non-linearity through the ReLU activation function, resulting in a more "discriminative" decision function.

3.2.2 Results

The results on the training and validation set are summarised in Figure 3.4. The loss curves show that the overfitting has been significantly reduced. The validation loss follows the same trajectory as the training loss up until epoch 15 when it starts to plateau until epoch 30. In the last 20 epochs, there is a slight upward trajectory of the validation loss. The slope of this increasing trend is considerably lower compared to the validation loss of VGG 1. Additionally, the variance of the validation loss has been noticeably reduced.

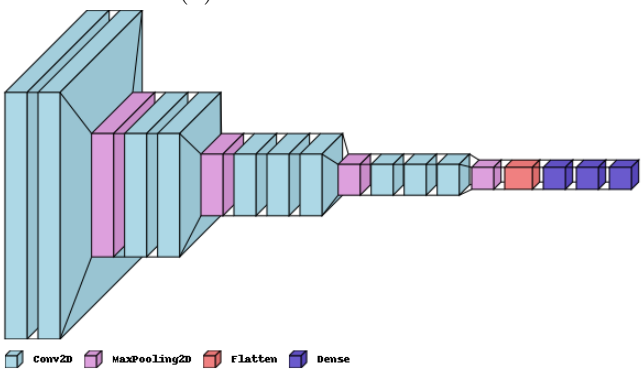
The overall performance of VGG 2 has also substantially improved compared to its shallower counterpart, with the validation loss after 50 epochs being less than half of what was obtained before. While the previous model attained a final validation loss of 0.7, this model never exceeds 0.5, fluctuating around 0.3.

Figure 3.3

(a) VGG 2 Architecture

Layer Name	Output size
Input (RGB)	(224, 224)
Pre-processing	(224, 224)
Conv3-16	(224, 224)
Conv3-16	(224, 224)
MaxPool2, /2	(112, 112)
Conv3-32	(112, 112)
Conv3-32	(112, 112)
MaxPool2, /2	(56, 56)
Conv3-64	(56, 56)
Conv3-64	(56, 56)
Conv1-64	(56, 56)
MaxPool2, /2	(28, 28)
Conv3-128	(28, 28)
Conv3-128	(28, 28)
Conv1-128	(28, 28)
MaxPool2, /2	(14, 14)
Flatten	25088
Dense	16
Dense	8
Output (Sigmoid)	1

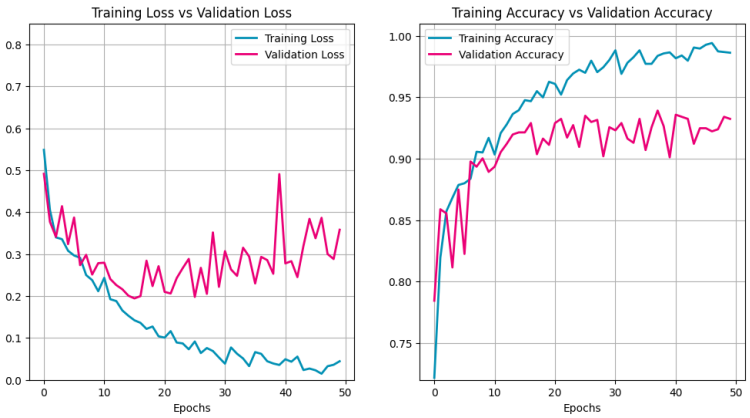
(b) VGG 2 Visualisation



The accuracy plot also confirms this observation, as this model correctly classifies 93% of unseen images in the validation set, achieving a 5% improvement.

We believe that these results are caused by the fact, that there are more subsequent convolutional layers present in this deeper version of the VGG architecture. Also, the introduction of the aforementioned (1×1) convolutions likely played a part in the performance increase.

Figure 3.4: VGG 2 results



Although VGG 2 does not outperform Sequential 2, the results are still impressive

given the fact that the model only contains convolutional layers and MaxPooling layers. We hypothesise that further improvements could possibly be made by considering the utilisation of BatchNormalisation and the He-Initialiser, which proved to yield significant performance boosts when implemented in the Sequential 2 model.

4 Residual Neural Networks

Deeper architectures play an important role in enabling convolutional neural networks to learn more complex patterns in each convolutional layer thus increasing their classification performance. However, He and Sun (2015) were able to show that a better model is not simply achieved by arbitrarily increasing its depth, as a model's accuracy tends to stagnate or even reduce for very deep architectures. The perhaps biggest breakthrough to overcome this limitation was the introduction of residual neural networks by He et al. (2016a). Their architecture made it possible to train models with incredibly high depth in a feasible time, resulting in significant accuracy improvements.

4.1 Residual Learning

The problem of utilising extremely deep architectures is not only the increased computational cost induced by a higher number of parameters but mainly the presence of a "degradation" effect, which causes saturation and degradation of the model's accuracy. He et al. (2016a) discuss that this degradation is not caused by overfitting, as a deeper architecture also negatively impacts the training error. Theoretically, adding more layers to a functioning model should not negatively affect the training accuracy, as the added layers could be constructed as identity mappings. To counter this problem, the idea is to let the network's layers learn the residual mapping instead of the original mapping. If $\mathcal{H}(x)$ is the desired underlying mapping of a layer, it is now fit to $\mathcal{F}(x) = \mathcal{H}(x) - x$ and then mapped back into $\mathcal{F}(x) + x$. He et al. (2016a) propose that residual learning works because it is easier to optimise the residual mapping. If in theory, the optimal mapping would be the identity function, it would be easier for the network to fit the residual mapping to 0 than to learn the identity mapping. This is partly because weights are usually initialised around a mean of 0, which would be close to the optimal residual mapping in that scenario, and because most solvers are not suitable for approximating the identity function.

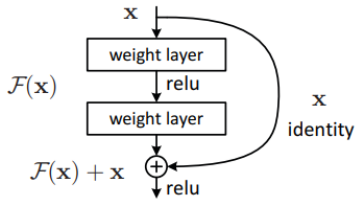
4.2 Implementation

Residual learning is achieved by utilising residual blocks and shortcut connections, as displayed in Figure 4.1. A residual block is composed of a few convolutional layers, whose goal is to learn the residual mapping. Each convolutional layer is followed by BatchNormalisation and a ReLU activation function. The identity mapping is applied by

adding the input tensor of the residual block to the output via the shortcut connection[1]. The job of each layer within a block is no longer to capture everything important about the input tensor that deserves to be passed along the subsequent layers but rather to understand what information it can add on top of the input.

Repeating this process several times ensures that each block has both a simpler task to learn and has access to better information to learn from, as the shortcut connections keep on passing the input forward. A further advantage is that the gradients will follow shorter paths, as they can also follow the shortcut connections during the backpropagation.

Figure 4.1: Shortcut connection by He et al. (2016a)



The implementation of residual blocks with shortcut connections is achieved through the utilisation of *identity blocks* and *projection blocks*. Identity blocks are used when the input and output dimensions of a residual block are identical. In this case, the shortcut connection can be executed directly, using pairwise addition[2] (Identity shortcuts are visualised by solid-line arrows in Figure 4.2b).

When input and output sizes differ, projection blocks are used, in which the dimensionality matching is achieved through (1×1) convolutions. When the tensor size is halved, the (1×1) convolutions are executed with a stride of 2 (Projection shortcuts are visualised by dashed-line arrows in Figure 4.2b).

4.3 ResNet14

4.3.1 Architecture

The first residual neural network we propose is ResNet14. Its architecture is displayed in Figure 4.2. Besides the introduction of the residual learning framework there are some other architectural changes from the previously discussed models. The first difference is the utilisation of a fairly large (7×7) convolution, followed by MaxPooling. Those two layers quarter the dimensions of the input tensor. The main part of the model is composed of 3 residual blocks with each residual block consisting of 2 convolutional layers. In contrast to our setup for Sequential 2, the BatchNormalisation is applied before, and not after the ReLU activation function of each convolutional layer.

[1] $\mathcal{F}(x) + x$ is performed by element-wise addition.

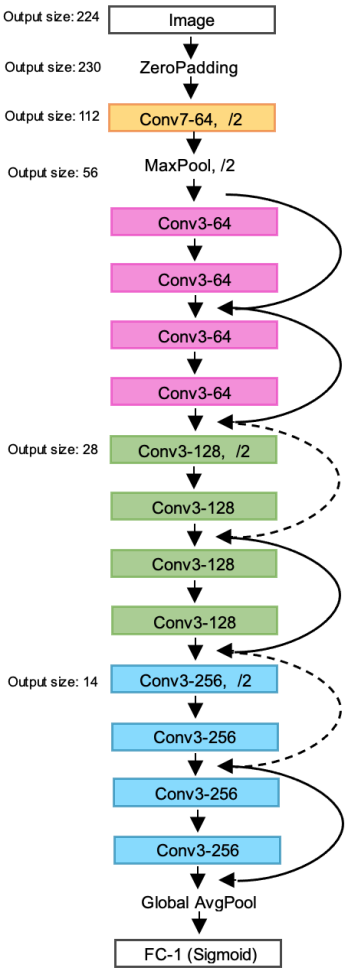
[2] We use addition for skip connections instead of concatenation to avoid too big of an increase in the number of parameters.

Figure 4.2: ResNet Architectures

(a) ResNet 14 and ResNet 32
BatchNorm and ReLU not displayed for brevity

Output size	**14-layer**	**32-layer**
112×112	7×7, 64, stride 2	
56×56	3×3 MaxPool, stride 2	
	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 3$
1×1	Global AvgPool, FC-1	

(b) ResNet14



The number of channels stays constant within each residual block but increases throughout the model.

One of the main differences is the fact that there are no MaxPooling or AveragePooling layers present within the residual blocks. The dimensionality reduction is thus achieved by a stride of 2 within the convolutional layers.

The last major change is the implementation of a Global AveragePooling layer, which was first introduced by Lin et al. (2013). Global AveragePooling is an alternative to fully connected layers, which were used in the VGG architecture and have a high risk of overfitting. Instead of vectorising (flattening) the last convolutional layers to connect them with the fully connected layers, Global AveragePooling generates a feature map for every category of the classification task, in our case 2. The average of each feature map is then taken and fed directly into the classification function, in our case the sigmoid

activation function.

With 14 layers and a total number of 2,785,281 trainable parameters, ResNet14 is our most complex model so far. Weights are initialised using the He-Normal initialiser.

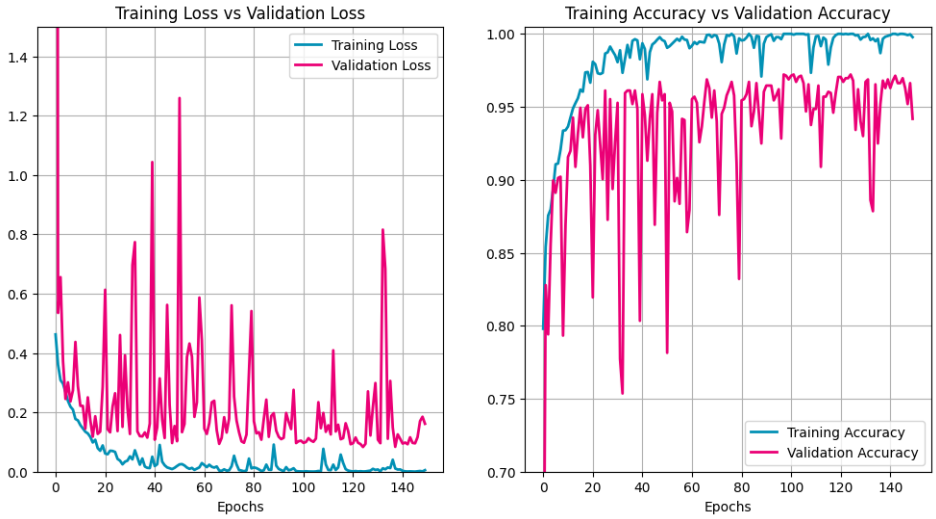
4.3.2 Execution Specifications

Due to the high number of parameters, this model is run for a total of 150 epochs. All other specifications remain unchanged.

4.3.3 Results

The results of ResNet14 are shown in Figure 4.3. Both training and validation loss are decreasing and there is no overfitting behaviour observable, which is a desirable characteristic. However, we can observe that the validation loss suffers from high variance. The volatility-spikes appear to be decreasing over time and the loss manages to stay below 0.2 for a considerable number of epochs, which is on par with the loss obtained by Sequential 2.

Figure 4.3: ResNet14 results



The high fluctuations are also visible in the validation accuracy, albeit the variance also seems to lower as more epochs are executed. It exceeds 95% accuracy on unseen data in several occurrences, even reaching accuracies above 96%, which is the highest accuracy achieved by any of our models so far.

Given the complex nature of this family of models, they may require a considerably higher number of epochs to be trained on, before obtaining stable and more robust outcomes. We further believe that deeper models like ResNet14 would significantly benefit from the fine-tuning of various hyperparameters. This could allow for both an increase in accuracy, as well as a more stable training run in general.

4.4 ResNet32

4.4.1 Architecture

ResNet32 introduces an even deeper architecture architecture than its predecessor, which is displayed in Figure 4.2a. It consists of 32 layers and has a total number of 5,224,833 trainable parameters, making it the most complex model in our entire analysis.

The main alterations to its shallower counterpart are the utilisation of 3-layer residual blocks and (1×1) convolutions, which were already introduced in our previous models.

Additionally, the last convolutional layer in each residual block now has a 4-fold increase in the number of channels.

4.4.2 Execution Specifications

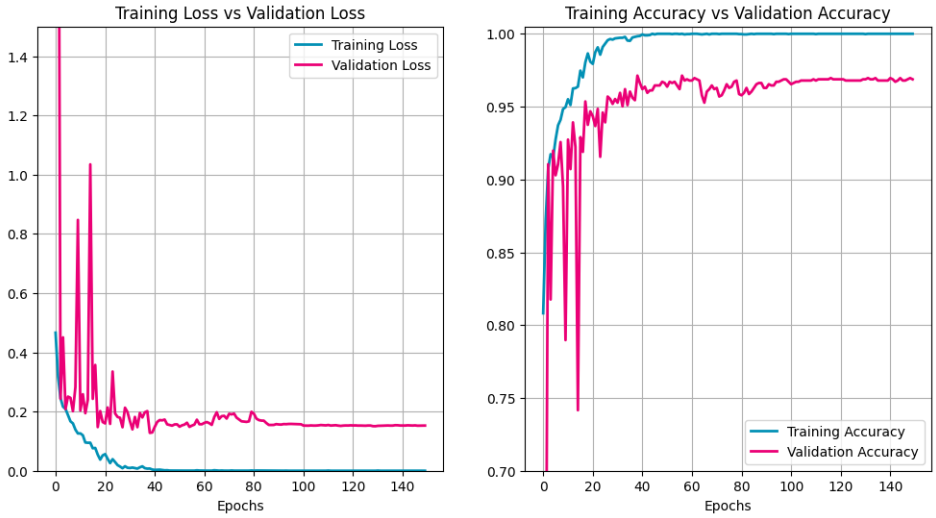
The model is run using a learning rate scheduler for the Adam optimiser. The learning rate starts at 0.0005 and then exponentially decays starting from epoch 15 after which the learning rate is calculated by the following formula:

$$lr_t = lr_{t-1} \times exp(-0.005)$$

Where lr_t denotes the learning rate at epoch t.

Although the Adam optimiser has the ability to adaptively alter its learning rate, the learning rate scheduler allows us to control the behaviour of the learning rate more accurately. The goal of the decaying learning rate is to stabilise the training run and to allow for finer tuning of weight updates in later epochs when the model has already learned its core features.

Figure 4.4: ResNet32 results



Additionally, Adam's β_1 parameter, denoting the decay of the 1st moment estimate, has been increased to a value of 0.999 (from its 0.99 default value) to guarantee a smoother training run with reduced variance. The model is run for 150 epochs.

4.4.3 Results

The results of ResNet32 are shown in Figure 4.4. The learning rate scheduler and the reduced starting learning rate in combination with the increased 1st moment decay rate yielded a significantly smoother training run. After an initial period of higher fluctuations, the variance starts to massively decrease from epoch 20 onwards for both validation loss and accuracy. After roughly 80 epochs the validation loss and accuracy plateau. The validation loss stays stable at a level below 0.2 and the validation accuracy stays above 96%, which are the best results achieved by any model in our experiments. With an appropriate training setup, the residual learning framework produces superior predictions in comparison to all previously discussed architectures.

5 Transfer Learning

Transfer learning describes the technique of taking a machine learning model which was pre-trained on a dataset for one task, and using it for a second related task on a different dataset. We consider a set of state-of-the-art neural network models, which were trained on large datasets for image recognition tasks, to investigate how they perform on our classification task.

The idea is, that the models have previously learned useful patterns when they were trained for their classification tasks. We use those patterns to then classify the images in our respective task. This approach is sometimes also referred to as feature extraction.

5.1 Models

The models we use for transfer learning have all been trained on the ImageNet dataset[1], which consists of over 1.4 million different images coming from 1,000 different classes[2].

Specifically, we consider the following 5 ConvNet models for feature extraction:

- Efficientnet B0 V2
- Efficientnet B3 V2
- ResNet50 V1
- ResNet50 V2
- VGG-19

The efficientnet architecture was introduced by Tan and Le (2019). Their main contribution was increasing the complexity of a model, not only by making it deeper (e.g. adding more layers), but by scaling all dimensions. They propose a method to scale the depth (e.g. number of layers), width (e.g. number of channels), and resolution (e.g. the input size of the images), using a constant ratio. This scaling method allowed for the creation of high-performing networks, with a relatively low number of parameters[3].

[1]The specific dataset can be found at: https://www.image-net.org/download.php

[2]Chihuahua is one of the classes among the 1,000, while Muffin is not. This, however, should not affect the models' performances in any form.

[3]While the ResNet50 has 26 million parameters, Efficientnet B0 has only 5.3 million parameters.

The residual neural network architecture was discussed in the previous chapter. The ResNet50 V1 by He et al. (2016a) and the ResNet50 V2 He et al. (2016b) both represent 50-layer deep residual neural networks.

The VGG-19 is the deepest VGG model introduced by Simonyan and Zisserman (2014) having 19 weight layers.

5.2 Execution Specifications

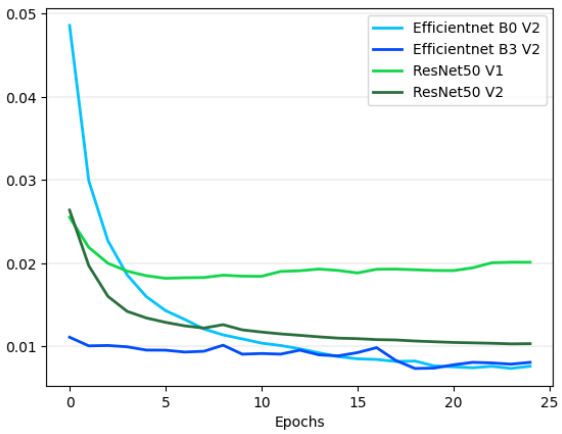
All models' weights have been obtained from a training run for a multiclass classification problem. We thus manually add a fully connected layer containing 1 neuron as the output layer to each model, using the sigmoid activation function, to obtain a prediction for our classification task which contains only 2 classes. This introduces a very small number of parameters that still have to be trained. However, this number is negligibly small in comparison to the number of pre-trained weights and parameters. Every model is executed for 25 epochs on the training set and evaluated on the test set. The image preprocessing has been performed using the respective recommended preprocessing functions for each model[4].

5.3 Results

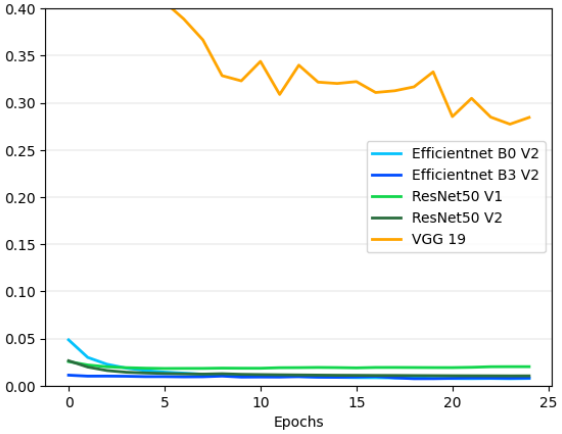
Figure 5.1 displays the models' losses on the test set. We can observe that all models have performed remarkably well, with the exception of the VGG-19.

Figure 5.1: Plot b) displays the same loss curves, with the inclusion of the VGG-19 and an adaption of the y-scale for better comparison.

(a) Loss Comparison - without VGG19



(b) Loss comparison - with VGG19



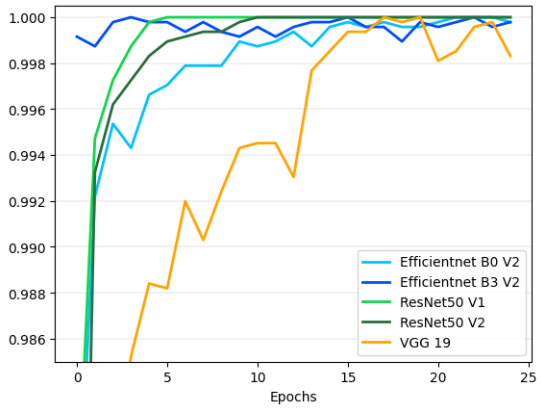
[4]Though the preprocessing procedures vary slightly from model to model, all preprocessing functions involve some form of centring or standardisation.

The Efficientnet B3 V2, which is the most recent of the considered models, arguably has the best performance, as it obtains a loss very close to 0 from the 1st epoch onwards. The other Efficientnet and ResNet models also achieve exceptional results, with loss values that are almost negligible in comparison to the losses we obtained from the models we manually built and trained.

The VGG-19 is the only model which scores a loss that is in the same range as the losses we experienced previously.

The predictive accuracy of all models also considerably outperforms the results we obtained from all previously discussed models. The VGG-19 is the worst-performing model, scoring an accuracy of slightly above 99% on unseen test images. All other models converge to 100% accuracy.

Figure 5.2: Transfer Learning Models Accuracies



These results show us how remarkably well some modern state-of-the-art models perform on unseen data. This performance is down to the fact, that the models are both more complex and advanced with respect to their architecture, but they have also been trained on a vastly larger training set for an incredibly high number of epochs[5].

[5]For instance, the original ResNet model was trained for 60×10^4 epochs.

6 Cross-Validation

We perform 5-fold cross-validation with respect to the zero-one loss, using the union of the training and test set. We evaluate the performance of the Sequential 2 model, as it was one of our best-performing models, while not being overly complex or deep. It thus yields a good trade-off between predictive accuracy and computational cost for training. The model is trained on 4 training folds for 50 epochs and evaluated on the remaining test fold. The results on the validation folds are shown in the table below:

Table 6.1: 5-fold cross validation

	Zero-One Loss	**Accuracy**
Fold 1	0.0725	0.9275
Fold 2	0.0481	0.9519
Fold 3	0.0473	0.9527
Fold 4	0.0338	0.9662
Fold 5	0.0658	0.9342
Mean	**0.0535**	**0.9465**

Sequential 2 performs very well on all folds and yields results lying in a very narrow range. This a desirable property, as it shows that the model's performance does not rely on which subset of the data it is trained or evaluated on, but that in fact, the model does a good job in predicting unseen data points in general. On average the model is able to classify 94.65% of unseen images correctly.

7 Conclusion

7.1 General Evaluations

The main insights to be drawn from our experiments are:

- Deeper architectures seem more capable of producing high-quality predictions.
- Multiple subsequent convolutional layers can improve results through the additional non-linearity of the activation function.
- (1×1) convolutions are a great way of adding more layers and thus more non-linearity to a model, without introducing too many more parameters and thus limiting the risk of overfitting.
- BatchNormalisation proved to be an extremely useful regularisation method, which significantly reduced the risk of overfitting.
- The He-initialiser is the preferred initialisation method, especially when dealing with deep architectures.
- Residual learning is a great way, of significantly increasing models' depths, avoiding the degradation effect and enabling training in a feasible amount of time.
- Using feature extraction by considering pre-trained state-of-the-art models yields incredible results with extremely low computational costs.

7.2 Potential Improvements

While the models described in this report provided valuable insights into the binary classification of images using neural networks, there are several areas for potential improvement. The biggest limitation we faced was caused by the computational resources at our disposal. With higher computational capacity we would have been able to consider a higher number of model setups, more advanced preprocessing methods, as well as more elaborate tuning and training phases.

One important way to improve our models' performances regards the training set size. During the preprocessing steps, we discussed how the techniques we applied had the effect of *replacing* the original images with the processed, augmented ones, rather than *enriching* them. In a scenario where we could enjoy more powerful computational resources, artificially enlarging the dataset could represent a solid and feasible strategy

to allow the models to better learn the patterns which distinguish the target classes, thus resulting in better generalisation and improved performance on unseen data. Generating more training pictures would have also allowed us to level the slight class imbalance between muffins (2,718 images in the entire data set) and chihuahuas (3,199 images in the entire data set).

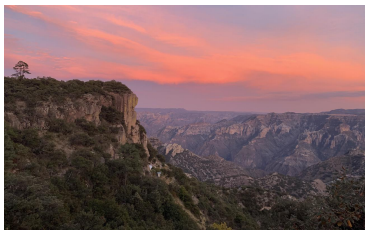
Furthermore, still concerning data preprocessing, a valuable way to perform image standardisation would be to apply it on a *per-pixel*-basis, as described by Krizhevsky et al. (2012), instead of the *per-image* standardisation we used.

A further area for improvement is hyperparameter tuning. Due to constraints on computational resources, hyperparameter tuning was conducted exclusively on the learning rate and only on one model, representing a pragmatic compromise. It is acknowledged that a more in-depth examination of multiple hyperparameters, as well as an application on more than one model, could yield additional insights and more robust hyperparameter estimations which potentially lead to further enhancements in model performances. We believe an in-depth hyperparameter tuning phase would especially benefit the training phase of the ResNet models, which require fine balancing in their setup to facilitate their full potential.

In addition, it is worth emphasising that the ResNet models often benefit from a longer training phase. Therefore, increasing the number of epochs for ResNet models can potentially offer improvements in the models' performances.

Investigating the content of the dataset, we observed a significant number of instances where the labelled class did not accurately represent the content of the images. This was especially the case for the chihuahua-class images, where the dataset contains many pictures of other dog breeds, humans, the Chihuahua region in Mexico or other unrelated content[1].

Figure 7.1: Examples of noisy data



The presence of irrelevant images within the dataset may hinder the model's overall performance, as it introduces noise from which the model tries to learn. To enhance the dataset quality, future efforts could involve a meticulous review of the dataset content.

Lastly, altering our models by further increasing their depth, using the scaling method proposed by Tan and Le (2019), could result in further performance increases.

[1]This is very likely caused by the fact, that the dataset has been obtained by scraping images from a Google search, which did not provide reliable results.

Bibliography

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

Declaration of Own Work

We declare that this material, which we now submit for assessment, is entirely our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of our work. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.