Program: Data Science for Economics
2023/24

Author:
Pietro Padovese

ID:
12356A

Course Coordinator:
Prof. Nicolò Cesa-Bianchi

May 6, 2024

Contents

1 Introduction

1.1 Task and Dataset

The task of this project requires constructing a binary classifier that can distinguish between muffin and chihuahua images, experimenting with different neural network architectures.
The dataset used is composed of a total of 5917 images, which are distributed as follows between train and validation set:

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

1.2 Preprocessing

Prior to being fed into the neural networks, the images underwent some preprocessing steps. Initially stored in JPEG format, they were resized to a uniform dimension of 224x224 pixels and interpreted in RGB color mode. Subsequently, the pixel values within these images, which conventionally range from 0 to 255, were rescaled to fit within a [0,1] interval. This process ensures constistency and aids in enhancing the efficiency of computational operations during model training and evaluation.

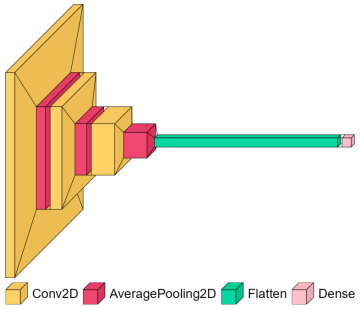
In addition, the images have been subject both to random flips and rotations. These techniques have been used to artificially augment the variety of data available during the training process, which can bring great benefit when working with relatively small dataset, like in this case. Doing so should help the model to learn more robust feature, invariant to small variations in the orientation of the images.

2 CNN-01

2.1 Architecture

The first architecture examined incorporates a basic convolutional neural network (CNN). After preprocessing, as detailed above, this model comprises three convolutational layers, each followed by an average pooling layer. The output of the final convolutional layer undergoes flattening and is then fed into a dense layer which return the desired output.
Below, each component is analysed in greater detail.

Figure 2.1: CNN-01 Architecture



Convolutional Layers

The goal of the convolutional layer is to extract features from the input data. It consists of a set of filters (or kernels) which are small-sized matrices that slide over the input data. Each filter detects specific patterns by performing a convolution operation. During this operation a dot product is computed between the filter weights and the corresponding input values. A process that is repeated across the entire image, producing a feature map that highlights the presence of different patterns or features.

All three convolutional layers of this network use a kernel size of 5x5 with padding, a technique that adds zeros around the edges of the image, in order to ensure that the spatial dimensions of the ouput feature maps remain the same as the input data.

The layers, on the other hand, differ in the number of filters applied. They have been structured with an increasing number of filters, starting with 16, then progressing to 32 and finally advancing to 64. The idea behind this choice is that as we move deeper into the network, the complexity of the features that the model can recognize increases, allowing the network to achieve a more refined understanding of the images.

The output of the convolution is fed into an activation function, specifically the hyperbolic tangent (tahn) function:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Its role is to introduce non-linearities into the model, increasing its flexibility and the capability to capture more complex relationships. The function outputs values in the range $[-1, +1]$, centered around 0, a property that can help the learning algorithm converge faster[1].

Average Pooling Layer

Each convolutional layer is followed by an average pooling layer with a kernel size of 2x2. These layers reduces each 2x2 patch to its average value, effectively downsampling it by a factor of 4. This operation not only reduces the computational costs, but also makes the features learned by the model more robust to position and orientation changes in the input image, while preserving the information from the input feature map.

Flattening and Dense Layer

Before passing the output of the last convolutional layer to a dense layer, the flatten layer reshapes the 3-dimensional feature maps into a 1-dimensional vector, so that it can be processed by a standard fully connected dense layer.

In the dense layer, each neuron in the previous layer is connected to a single node. Its output is then processed by a sigmoid function, that squashes it within the $[0, 1]$ range, representing the probability of the input image belonging to one of the two classes.

2.1.1 Compiling

Before starting with the training process, we need first to configure the model specifying various parameters that govern how the model learns and how it optimizes its performance.

Loss function

The loss function quantifies the distance between the model predictions and the actual labels. It serves as feedback to the optimizer to guide its process of adjusting the parameters of the model.

A common choice in binary classification problem, and the one adopted in this project,

[1] Lecun, Yann & Bottou, Leon & Orr, Genevieve & Müller, Klaus-Robert. (2000). Efficient BackProp.

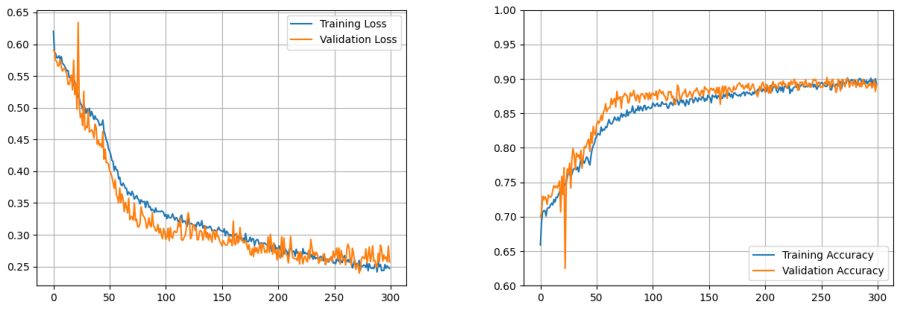
is the binary cross entropy loss function. This function captures the confidence of the predictions, which provides more information than just class labels.

Optimizer

The purpose of the optimizer is to adjust the parameters of the model iteratively during the training in order to minimize the define loss function. The chosen optimizer was the Adaptive Moment Estimation (Adam) [2]. This algorithm can be though as an "extension" of the Gradient Descent, which minimize the loss function by moving the weights in the direction of the steepest descent of the function's gradient. The size of this steps is determined by the learning rate, which determines how much we adjust the parameters. Traditional gradient descent methods use a fixed learning rate for all parameters, and remains constant for the whole training process. With Adam instead individual learning rates are used for each parameter and are updated at each iteration using the first and second moment of the gradients. The first moment helps in smoothing the optimization trajectory by accumulating past gradients, the second allows to adjust the rates based on the variance of the gradient, allowing parameters with large gradients to have smaller updates and vice versa.

2.1.2 Training and results

Figure 2.2: CNN-01 Results



(a) Loss

(b) Accuracy

The model has been trained for 300 epochs and the performance on both the training and validation set are shown in Figure 2.2. At first glance, one particular aspect that emerges from the graphs is that over a long period of training, performance on the validation set was better than on the training set. One possible explanation for this unexpected behavior may be the rotation and flipping steps to which images are subjected. Indeed, these steps are only applied in the training phase and not during validation, as

[2]Kingma, Diederik P. & Ba, jimmy. (2015). Adam: A Method for Stochastic Optimization

these changes would alter the data whose purpose is to provide a scenario as realistic as possible. These alternations actually make it more difficult to learn from the training dataset, but at the same time they help the model learn more general features, which make for better performance on images not previously seen by the algorithm.

Beyond that, the model shows a slow convergence toward an optimal solution. This could be due to various reasons, including a starting learning rate that is too small; the main hypothesis, however, is that the model is too simplistic to capture the various patterns needed for image recognition.

Overall, for most of the epochs the validation loss stays between 0.25 and 0.3. With regard to validation accuracy, the mode in the final stages of training stands at values just under 0.9

Since this model was more of a starting point, the next steps included expand its structure, adding more layers and changing some of the choices made in its construction.

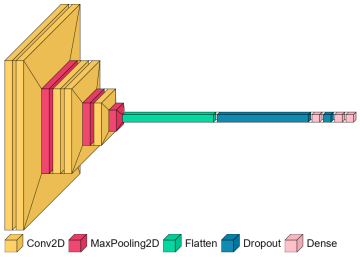
3 CNN-02

3.0.1 Architecture

The first major change in the structure of the model is the depth of its layers. Now, instead of 3 convolutional layers, in total there are 5, with a large number of filters too, respectively 2 layers with 32 filters, 2 with 64 and a last one with 128.

In addition, the two layer with 32 filter and the two layers of 64 are stacked together forming two convolution blocks, instead of being separated by pooling layers. This causes the inputs in the second layer to be the features extracted from the first one, without them being pooled, making the extracted features of the second a "higher level" than the previous.

Figure 3.1: CNN-01 Architecture



Two dense layers with 512 and 128 nodes were also added after the flattening operation. These two layers can enhance the feature aggregations capabilities of the model. After extracting complex features with the convolutional layers, they are aggregated and combined in the dense layer. Having more dense layer means increasing the number of non-linear transformations to the feature space that the model can capture.

Overall, these changes bring the total layers from 3 to 7, with the number of parameters increasing from 115.489 to 51.587.233, making the second model much more complex and flexible than the first.

ReLU

Another modification implemented pertains to the activation function. Now, the output of the convultional layers, before being passed to the subsequent layer, is transform by the ReLU (Rectified Linear Unit) activation function.

$$f(x) = max(0, x)$$

In contrast with tahn function, which maps input value to the range $[-1, +1]$, the ReLU outputs the input if it is positive and zero otherwise. This not only improve the computational efficiency by introducing sparsity in the network, but also tackles the problem of diminished gradients, which may emerge in a deeper model like this one. This problem refers to the fact that during the backpropagation process, since the gradients are computed through the chain rule of calculus from the output layer, backward through the network, the repeated multiplications that happens in each layer may lead the gradient to diminish significantly in magnitude, slowing down the process of training. For how the ReLU is constructed, multiplying for its gradient is equivalent to a multiplication by 1 when the input is positive and zero otherwise, meaning that it will either stays as it is or become exactly zero.

The choice to use the ReLU activation function has also led to reflections on the weight initialization process. The optimization path of adjusting the weight associated to each parameter, needs indeed a starting point, a choice that can strongly affect the training process. Although randomly drawing weights from a distribution in a specific range is a possibility, more tailored systems have been developed that consider the type of activation function used.

In the previous mode, which used the tahn activation function, the Xavier uniform initialization was selected. This method, which has been proposed by the authors in 2010[1], when the most common choice was the tahn, revolves around the idea of balancing the variance of the outputs neurons across all layers to limit the problem of vanishing/exploding gradients. To achieve this in each layer the initialization try to keep the variance of the input and output gradient the same. The weights are initialize drawing from a normal distribution with mean zero and standard deviation $\sqrt{6/n_{in} + n_{out}}$, where n_{in} is the number of input units in the weight tensor and n_{out} is the number of output units. This initialization is effective for activation functions that do not distort the variance significantly, which is not the case with the ReLU.

A method that overcomes this limitations is the He initializer[2] , which goal remains the same, namely to control the variance of the gradients, with a slightly different approach. Since the ReLU reduces the negatives value to zero, halving the variance, He initializer compensates for this reduction by doubling the scale of the weight's variance. In this sense the weights are drawn from a normal distribution distribution with mean zero and standard deviation $\sqrt{2/n_{in}}$. This initializer helps in avoiding the dead neurons problem, which happens when a neuron outputs zero for all inputs. They can occur

[1]Xavier, Glorot & Yoshua, Bengio (2010). Understanding the difficulty of training deep feedforward neural networks.

[2]He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian (2015). Devling Deep into Rectifiers: SUrpassing Humal-Level Performance on ImageNet Classification.

when the initial weights are too small. The He initializer should ensure that the initial weights are larger enough to keep the majority of neurons active.

BatchNormalisation

Between the convolutional layer and the activation function a regularisation method has been added. Now the feature maps undergo the process of BatchNormalisation, which helps to improve training stability and speed up the convergence. During the training process, the distribution of activations within a neural network shifts, a problem known as internal covariate shifts. This can cause the optimization algorithms some problems since it needs continually to adapt to the changing distribution. By normalizing the activations of each layer within a mini-batch, we ensure that the distribution of the activations remains stable during training.

MaxPooling

The pooling method has been changed too. The average-pooling layers have been substituted by max-pooling ones. While retaining the benefits derived from downsampling, max-pooling operation by selecting only the most "active" features has a better chance of discarding noise and less relevant signals within each pooling patch. Emphasizing more the presence of specific features regardless of their precise location within the region, it can lead to simpler decision boundaries, making the learning process easier.

Dropout

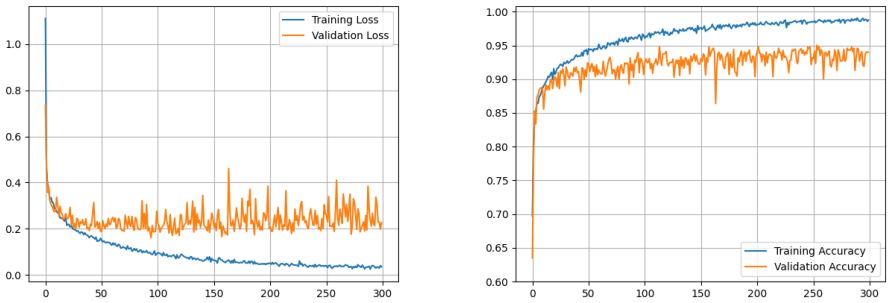
Finally, one dropout layer has been added before each of the new Dense layer. Each of them randomly sets a 20% of neurons to zero. By reducing the possibility of neurons to co-adapt, the network it is forced to learn more robust features[3]. Overall, this regularization technique can improve the robustness and generalization ability of the neural network.

[3]Krizhevsky, Alex & Sutskever, Ilya & Hinton, Geoffrey E. (2012). ImageNet Classification with Deep Convolutional Neural Netowrks.

3.0.2 Result analysis

The results shown in Figure shows that the model converge much faster than the previous architecture and signs of overfitting already emerge around the 50th epoch. It reaches better results both in terms of validation loss and accuracy, since the loss reaches lows below 0.2 and the accuracy usually stays around the 0.90-0.95 range. Despite the presence of many fluctuations in the results, this model has been judged as an improvement from the previous version, and as this some of the changes made for this model will also be incorporated into the next structure analysed.

Figure 3.2: CNN-02 Results



(a) Loss

(b) Accuracy

4 Residual Learning

4.1 Architecture

For the third model a different approach has been taken regarding the architecture of the network. This new implementation revolves around the concept of residual learning, introduced with ResNet model by He et al[1]. This framework aims at easing the training of deep neural networks by introducing skipping connections within network layers. If in traditional CNN architectures the aim is to transform the input data into a representation progressively closer to the desired output, in residual learning the focus shift to learning residual function that capture the discrepancy between the desired output and input to a particular layer.

The core element in this architecture is the identity/convolutional block (with the difference being whether or not the input activation has the same dimension as the output). Within this block, the input will follow two paths. The "main path", which is a standard CNN flow, and the "Skip connection path",that allows the input to bypass this stream. If we denote the input as X, and the desired mapping as $H(X)$, we are basically letting the network fitting $F(X) = H(X) - X$, the residual mapping. The output of the fitting and the initial input are finally added together and passed to the activation function. Since operating on the residual it is easier than figuring out from scratch the entire input-output mapping, this technique should help our model to converge faster.

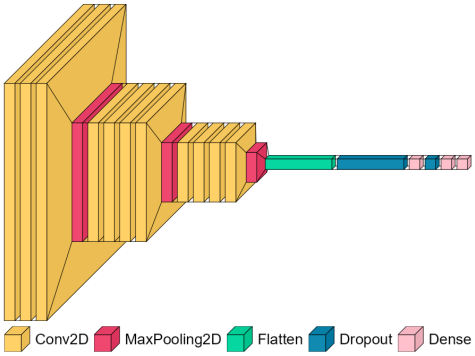
We take advantage of the ability of residual learning to manage more efficiently deeper neural network by adding many layers to our model. Overall, the structure is composed in this way:

- A first convolutional layer of 16 filter.
- An identity block of two convolutional layers with 16 filters.
- A convolutional block of three convolutional layers with 32 filters.
- A convolutional blcok of three convolutional layers with 64 filters.

[1]He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing, & Sun, Jian (2015). Deep Residual Learning for Image Recognition.

What is carried over from the previous model are the choice of using ReLU as activation function, the implementation of BatchNormalisation, the addition of the dense and dropout layers.

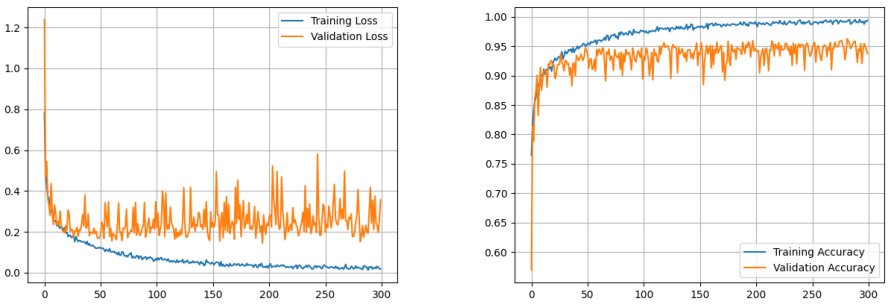
Figure 4.1: Res. Learn. Model Architecture



4.2 Result analysis

The new architecture shows results not too dissimilar from the second model. Although it achieves slightly higher performance peaks than the previous version, the problem of fluctuations has remained, and even seems to be accentuated, making it difficult to determine which of the two architecture is best for this task. In the next section, both models will be reviewed by cross validation to obtain more robust risk estimates for comparison.

Figure 4.2: Res. Learn. Model Results



(a) Loss

(b) Accuracy

5 Cross-validation and Conclusion

5.1 Cross-validation

In this final step, the CNN-02 and Residual Learning models were evaluated using a 5-fold cross-validation approach to estimate the risk with respect to the zero-one loss. The whole dataset was utilized in creating the 5 folds, thereby merging the training and validation sets. Consequently, for each of the 5 splits in the training process 4/5 of the entire dataset were employed for training, while the remaining 1/5 was reserved for testing. Given the computational intensity of this process, the models were trained for 50 epochs during each iteration. To perform evaluation on the validation fold, the weights of the model achieving the best result in terms of training accuracy across the epochs were utilized.

Table 5.1: 0-1 Losses

Iteration	**CNN-02**	**RS**
1	0.086	0.082
2	0.067	0.075
3	0.056	0.077
4	0.070	0.080
5	0.073	0.098
Avg. Loss	**0.0704**	**0.0824**

The results shown in Table 5.1 demonstrate that the CNN-02 outperforms the Residual Learning model, achieving a lower average loss.

5.2 Conclusion

Despite the outcome that might lead to considering the residual learning framework as less efficient for this specific task, the difference is not so significant as to exclude it completely.

What emerged more strongly from this analysis is the benefit in terms of convergence

speed and overall performance derived from transitioning from the first to the second model. A deeper architecture, the choice of activation functions along with weight initialization, and the utilization of normalization techniques such as dropout and batch normalization have proven beneficial to our goal.